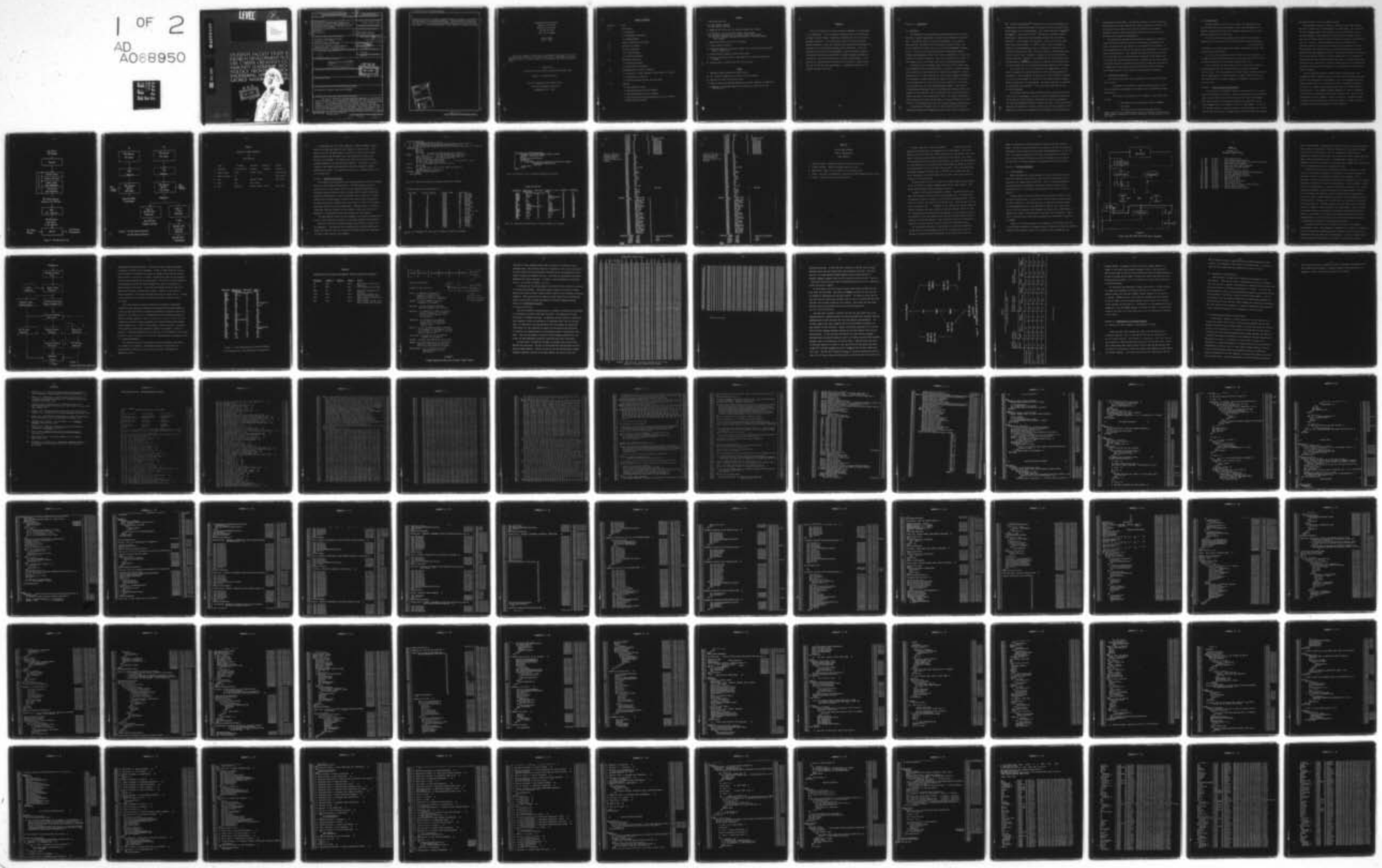
AD-A068 950 GEORGE WASHINGTON UNIV WASHINGTON D C SCHOOL OF ENGI--ETC F/G 9/2
DEVELOPMENT OF EXPERIMENTAL COMPILERS TO GENERATE EMULATORS FOR--ETC(U)
APR 79 R E MERWIN DASG60-78-C-0115

UNCLASSIFIED NL

1 OF 2
AD
A068950



2 SC

THE
GEORGE
WASHINGTON
UNIVERSITY

STUDENTS FACULTY STUDY R
ESEARCH DEVELOPMENT FUT
URE CAREER CREATIVITY CO
MMUNITY LEADERSHIP TECH
NOLOGY FRONTIE ESIGN
ENGINEERING APP ENC
GEORGE WASHIN NIV

SCHOOL OF ENGINEERING
AND APPLIED SCIENCE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER -	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) 6 Development of Experimental Compilers to Generate Emulators for the BMD DDP Test Bed From High Level Language.		5. TYPE OF REPORT & PERIOD COVERED Final, 8/78 to 3/79
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) 10 R.E./Merwin		8. CONTRACT OR GRANT NUMBER(s) 15 DASG 60-78-C-0115
9. PERFORMING ORGANIZATION NAME AND ADDRESS The George Washington University Washington, D.C. 20052		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 12 125p
11. CONTROLLING OFFICE NAME AND ADDRESS The Ballistic Missile Defense Advanced Technology Center Huntsville, AL		12. REPORT DATE 11 1 April 1979
		13. NUMBER OF PAGES 102
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) 9 Final rept. Aug 78-Mar 79,		15. SECURITY CLASS. (of this report) Unclass.
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Microprogram, Compiler, High Level Language

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The use of compilers to go from high level langauges to microprograms is not commonplace. A few experimental compilers have been described but no support of this concept is provided by manufacturers of user microprogrammable computers. In this feasibility study two existing compilers accepting dialects of Pl/1 were considered for generating microprograms for computers with a horizontally encoded control word format. One of the compilers which accepts the PLM language as input and produces quadruple intermediate text

next page

153 370 [illegible]

formats and DEC PDP 11 assembly language statments as outputs, was selected for modification to produce micorprograms. A post processor is described which was added to this compiler to produce microcode for the DEC PDP 11/45 minicomputer. A performance analysis of this compiler concludes the report.

UNCLASS

Development of Experimental Compilers to Generate Emulators for the BMD DDP Test Bed From High Level Languages

Final Report
April 1, 1979

Sponsored by

The Ballistic Missile Defense Advanced Technology Center

Contract No. DASG60-78-C-0115

School of Engineering and Applied Science

The George Washington University

Washington, D.C. 20052

TABLE OF CONTENTS

FIGURES

TABLES

ABSTRACT

The use of compilers to go from high level languages to microprograms is not commonplace. A few experimental compilers have been described but no support of this concept is provided by manufacturers of user microprogrammable computers. In this feasibility study two existing compilers accepting dialects of PL/1 were considered for generating microprograms for computers with a horizontally encoded control word format. One of the compilers which accepts the PLM language as input and produces quadruple intermediate text formats and DEC PDP 11 assembly language statements as outputs, was selected for modification to produce microprograms. A post processor is described which was added to this compiler to produce microcode for the DEC PDP 11/45 minicomputer. A performance analysis of this compiler concludes the report.

Section 1.0 INTRODUCTION

1.1 BACKGROUND

The techniques of microprogramming provide facilities for the implementor of a computer application to optimize the performance of the hardware. Calculations which are carried out repeatedly, e.g. operating system kernels, and digital filter and graphics display routines, can all be speeded up by factors of up to ten with microprogrammed implementations. In spite of the potential performance enhancements possible through microprogramming, there hasn't been a widespread use of these techniques by the users of computers. Much of this reluctance on the part of the computer user is the inherent difficulty of writing microprograms.

Although high level languages (HLL) are widely used to relieve computer programmers from writing applications in machine or assembly language, this facility hasn't been developed to simplify the preparation of microprograms which is a more difficult task than assembly language programming. A few attempts to develop experimental compilers from a HLL to microcode have been described (1,2) but there has been no adoption of this concept by the manufacturers of "user" microprogrammed hardware. The concepts of designing such compilers haven't been extensively developed and the preparation of microcode from a HLL for computers with a horizontal encoded control word hasn't been accomplished to date. This report will describe some techniques which have been developed to establish feasibility of generation of microprograms from a HLL for a range of computer architectures.

The production of support tools to better utilize the facilities offered the application programmer has been a major object of research at

GWU. Abd-Alla and Karlgaard(3) developed one of the first algorithms for automatically optimizing microcode by scanning the object code and identifying segments to be microprogrammed. This work was extended by Moffett(4) and Evans(5). More recently Fodor(2) implemented a microcode compiler based upon the high level microprogramming language, MPL, proposed by Eckhouse(1). This research had two objects: one was to see if a translator writing system (TWS), i.e. XPL(6), could be used to implement a compiler for microprograms;and second to determine the efficiency of a compiler developed using a TWS. From this research it was determined that the TWS could be used to implement a compiler for a HLL which directly produces microcode. Further it was demonstrated that the resultant microcode produced by the MOD 3 compiler was nearly as efficient in storage utilization and running time as the equivalent microcode produced by hand coding.

In the later sections of this report the term host machine will be used to designate the computer hardware for which the microprograms are implemented. This terminology is widely used to differentiate the hardware in which the microprograms are executed and the computer machine language being interpreted by the microprograms. The machine language is considered as defining the target machine. In this study the target machine will be the PLM language.

The Ballistic Missile Defense (BMD) distributed data processing (DDP) test bed has a requirement to provide a flexible computational environment and the DEC VAX 11/780 computers selected for the test bed incorporates the user microprogramming feature to support this need. Since direct (hand) generation of microcode is tedious, error prone, and time consuming, it was apparent that some way of simplifying the preparation of microprograms for these machines was desirable. The DEC VAX 11/780 has a 96 bit horizontally encoded control word and a feasibility study to determine if microcode could be produced for this machine from a HLL was initiated at The George Washington University where experimental compilers of this type (2) had previously been designed and implemented.

Two experimental compilers were available to support this feasibility study and the main effort was devoted to activating and evaluating each system to determine its applicability to the generation of microcode. It was determined that only one of the compilers was satisfactory in a selection process described below. This compiler was modified to produce microcode for the DEC PDP 11/45 as a substitute for the DEC VAX 11/780 machine since the control word field descriptions weren't available for the latter machine. The modified compiler has demonstrated an ability to generate microcode for a computer with a horizontally encoded control word as described below. A performance analysis of the compiler shows acceptable performance.

1.2 EXPERIMENTAL OBJECTIVES

The following objectives were established for this feasibility study at the outset:

(a) Modify the two available compilers described above so that they will produce microprograms for a computer with a horizontally encoded control word.

(b) Evaluate the performance of the compiler. Two criteria will be used:

1. The number of micro instructors needed to implement test cases.
2. The number of main storage references required.

These criteria will be compared between microcode produced from FORTRAN and DECPDP Assembly Language and microcode compiled by the quad compiler for the test cases.

1.3 ACCOMPLISHMENTS

(a) Both compilers which were to be used in the feasibility study were activated and test cases prepared. After some investigation it was decided to concentrate on the PLM to quadruple compiler due to its much greater flexibility at the HLL level.

(b) A post processor was developed which converts the quadruples generated by the quad compiler into special register oriented quadruples. The post processor is designed to be general purpose and capable of generating register quads for any machine architecture.

(c) A micro code generator for the DEC PDP 11/45 was implemented. It is again a generalized design capable of converting R quads into microcode for any host machine with a horizontal encoded control word.

(d) An evaluation was made relative to the criteria of the number of microinstructions and main storage references required to implement the test cases. The microinstructions and main storage references generated by the quad compiler are compared against the same criteria derived from hand coded DEC PDP 11 assembly language, and FORTRAN IV implementations of the test cases.

Section 2. COMPILER DESIGN AND DESCRIPTION

As noted above, two available compilers were considered for the generation of microprograms from a HLL. Both were implemented using a translator writer system (TWS) (6) which is based upon the XPL high level language. This system provides a methodology for producing a compiler for a high level language which can be defined in terms of a BNF grammar. It consists of a language analyzer, a prototype compiler (called skeleton), and compiler (called XCOM) which accepts a program written in the XPL high level language and produces object code for the IBM 370 system.

The procedure for using the TWS is to define the high level language, for which a compiler is to be implemented, in a BNF format. This is entered into the language analyzer which produces a set of parsing tables for the specified language. These tables are entered into the prototype compiler and a set of routines written in XPL are inserted to carry out the compilation activity. These are: semantic routines to define the meaning of the individual phrases of the HLL input statements; code generators to convert these meanings into object code for the specified host machine; and symbol table and other data management routines. The resultant program is a compiler for the specified HLL written in the XPL language which is now compiled by the XCOM compiler into an IBM 370 object load module. This load module when loaded into a 370 computer will compile a program written in the specified HLL into object microcode for host machine. The operation of the TWS is illustrated in figure 1.

The two compilers accepted a variant of PL/1 as an input HLL. One designated the MOD 3 Compiler accepts a version of MPL (1) and produces microcode for the Interdata MOD 3 minicomputer. The second designated the Quad Compiler accepts PLM (7), a subset of PL/1 which is supported by INTEL as a HLL input to a compiler which produces object code for the 8080 series of microprocessors. This compiler produces quadruples, a form of intermediate text (8) which are translated into DEC PDP 11 machine code. An overview of the operation of these compilers is given in figure 2. A detailed program listing of the Quad compiler can be found at appendix 7.1,Table I provides a definition of the quads generated by the Quad Compiler.

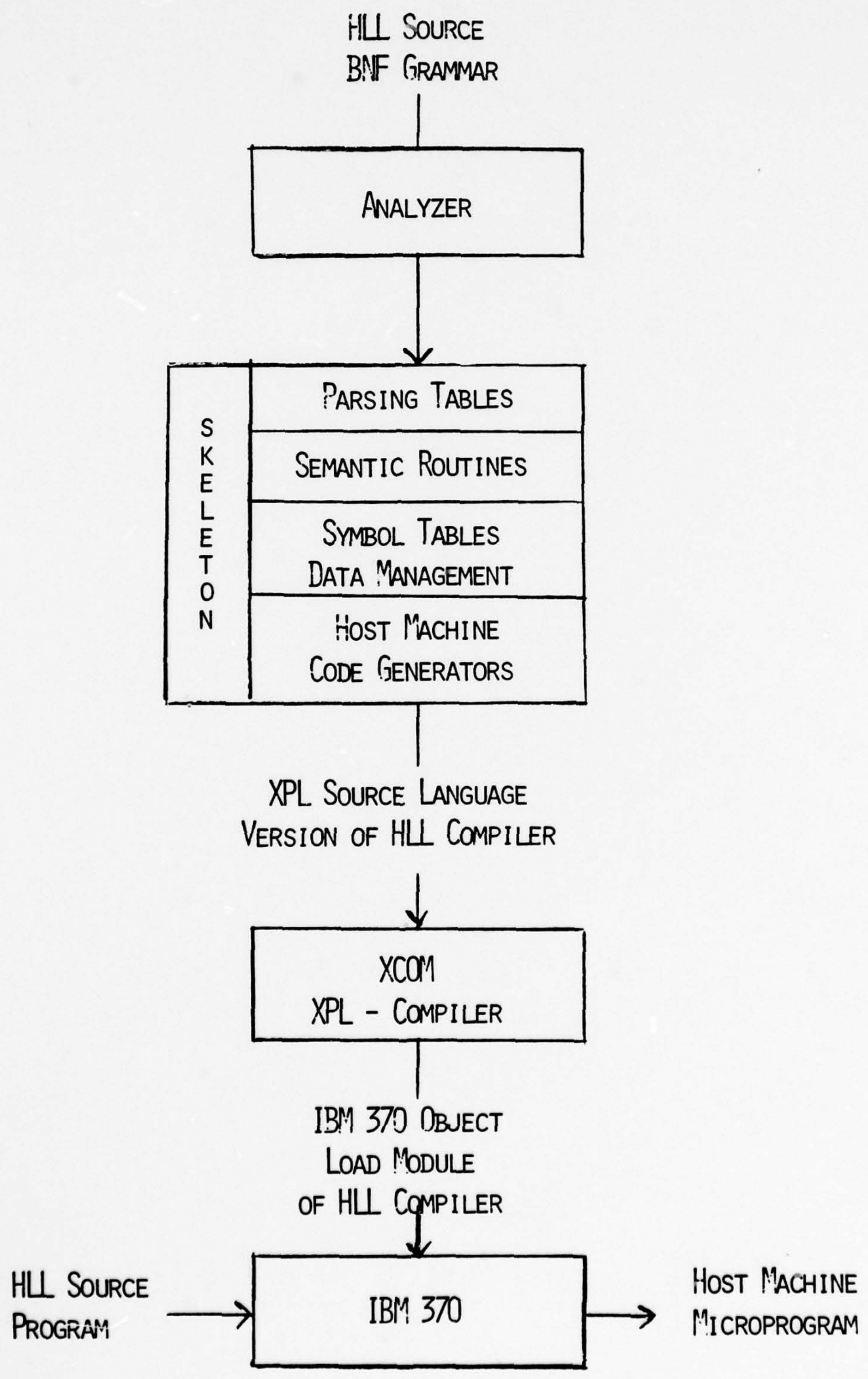


Figure 1. TWS Functional Flow

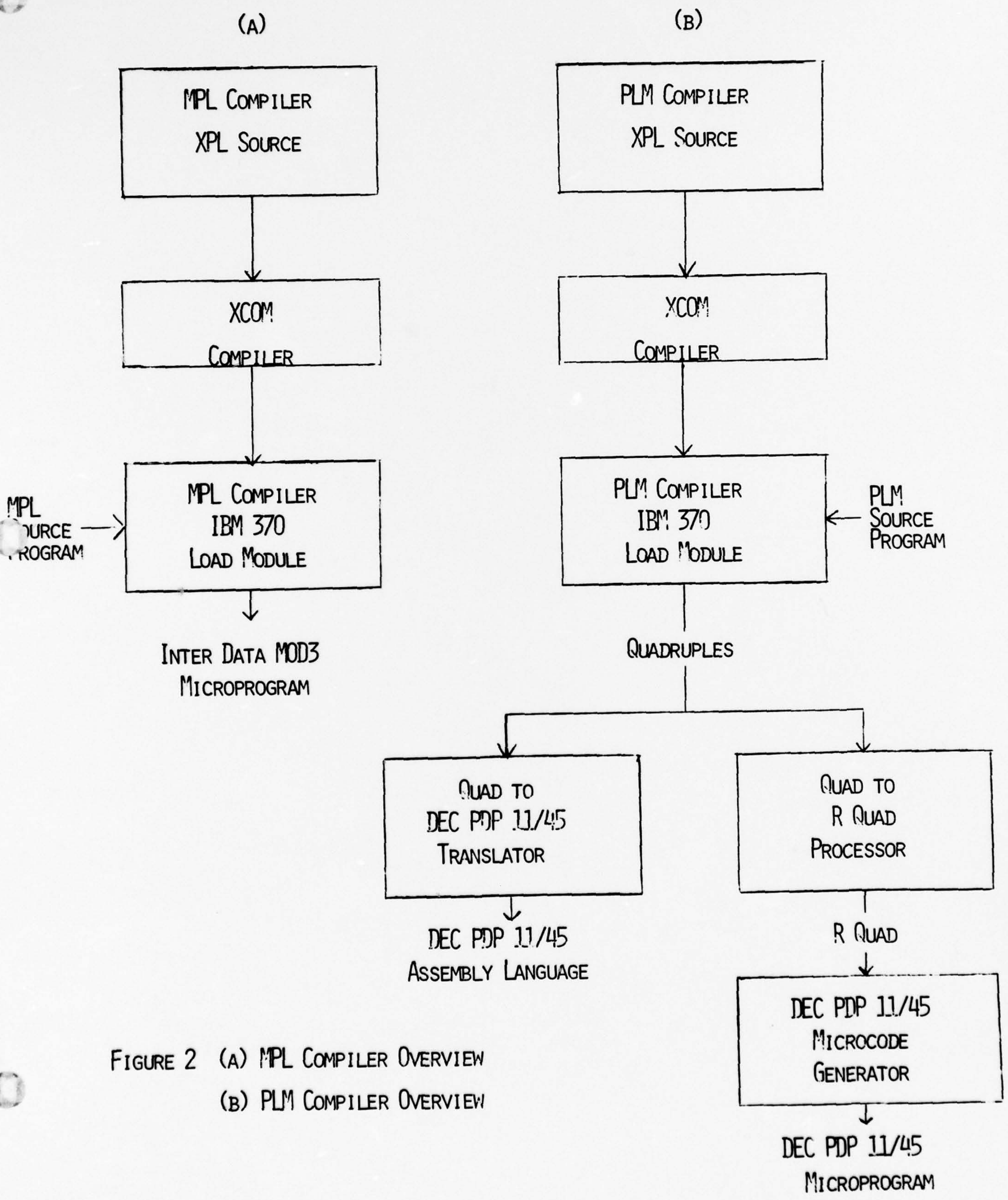

(A)
MPL COMPILER XPL SOURCE
XCOM COMPILER
MPL SOURCE PROGRAM
MPL COMPILER IBM 370 LOAD MODULE
INTER DATA MOD3 MICROPROGRAM
(B)
PLM COMPILER XPL SOURCE
XCOM COMPILER
PLM COMPILER IBM 370 LOAD MODULE
PLM SOURCE PROGRAM
QUADRUPLES
QUAD TO DEC PDP 11/45 TRANSLATOR
DEC PDP 11/45 ASSEMBLY LANGUAGE
QUAD TO R QUAD PROCESSOR
R QUAD
DEC PDP 11/45 MICROCODE GENERATOR
DEC PDP 11/45 MICROPROGRAM

FIGURE 2 (A) MPL COMPILER OVERVIEW
(B) PLM COMPILER OVERVIEW

TABLE I

QUADRUPLE FORMATS GENERATED

BY

QUAD COMPILER

	Type	Operator	Operand 1	Operand 2	Result
1.	Arithmetic	+,-,*,GT,LT,EQ	Variable	Variable	Variable
2.	Branch (Cond)	BT,BF	Variable (Flag)	-	Addr (Label)
3.	Branch (Uncond)	BR	-	-	Addr (Label)
4.	Data	ASGN	Variable (ADDR)	-	Addr (Var)
5.	Label	LAB	Addr (Label)	-	
6.	Array	SUBS/SUBL	Address (Array)	Index	Addr (Var)

To illustrate the use of these compilers, a sample program is shown in terms of its input and output for each compiler in figure 3 and 4. Note that the input to the MOD3 compiler is written in terms of the internal machine registers and functional units even though the statements are written in a HLL format. This implies the user of this HLL must have an intimate knowledge of the Interdata MOD3 hardware. The input to the quad compiler is more general and appears much more like statements normally associated with a HLL. This generality does lead to a complexity which will be discussed below.

Section 3. <u>SELECTION OF APPROACH</u>

Both compilers were activated at the George Washington University Computer Center (An IBM 370 MOD 148 Installation). The MOD3 compiler was easily activated but the quad compiler which had been originally implemented on a different computer required a number of changes before becoming operational. During the implementation of the test programs shown in table II, it became apparent that the MOD3 compiler suffered from a serious lack of generality. As noted above, the HLL statements had to be written in terms of the internal functional components of the Interdata MOD3. Programs written in a machine independent form were rejected by this compiler. Based upon this lack of generality, it was decided not continue working with this compiler.

The quad compiler accepts as input programs a very natural formulation of the test case algorithms and produces an equivalent quadruple representation along with a DEC PDP 11 assembly language version which is derived from the quadruples. Some problems were encountered and solved for handling of arrays and nested DO loops and a "built in" multiply routine was implemented to permit a wider range of test programs.

```
GREATEST_ELEMENT:
            PROCEDURE OPTIONS (MAIN):
  /*   MPL PROCEDURE TO FIND THE GREATEST INTEGER IN A LIST */
            DCL (R0,R1,R2,R3,R4,R5,R6,R7,AR,DFR) BIT (8):
  /*   R0 AND R1 CONTAIN POINTER TO LIST. R7 IS USED TO INDEX THE LIST */
INITIALIZE:
            R5 = 0:
            R4 = 0:
            R7 = 0:
            R6 = 49:    /* LIST IS ASSUMED TO BE 50 LONG.*/
READ:                   /* READ UP NEXT OPERAND IN LIST */
            MAL = R7+R1:          /* USE R1 AS BASE ADDRESS */
            MAH = R0 + CARRY:   /* DOUBLE PRECESION ADD */
            MDR = MS(MAR):       /* LOAD OPERAND   */
            IF R5 <_MDH THEN GOTO SWAP:
            IF R5 ¬= MDH THEN GOTO NEXT:                 /* SHOULD BE  ¬= */
            IF R4 > MDL THEN GOTO NEXT:
SWAP:                /* PLACE MDR IN R4 AND R5 */
            R4 = MDL:
            R5 = MDH:
NEXT:           /* GET NEXT ELEMENT AND CHECK IF DONE */
            R7 = R7 + 1:
            IF R7 < R6 THEN GOTO READ:
FINISH:     GOTO GREATEST ELEMENT:
            END:
    EOF EOF EOF
```

Fig 3 (a) MPL Source Program for Greatest Element Test Program

```
OBJECT LISTING FOR INTERDATA III

  REF LINE        MICRO LOCATION        OBJECT CODE        MNEMONICS
    8                   0                  5500            LI    R5."00"
    9                   1                  5400            LI    R4."00"
   10                   2                  5700            LI    R7."00"
   11                   3                  5631            LI    R6."31"
   12                   4                  4873            L     AR.R7
   12                   5                  CC17            A     MAL.R1
   13                   6                  CB0F            A     MAH.R0
   15                   7                  3100            C     MEM.READ
   15                   8                  4853            L     AR.R5
   15                   9                  E897            S     AR.MDH
   16                  10                  1111            B      L I. 17
   16                  11                  4853            L     AR.R5
   16                  12                  E897            S     AR.MDH
   17                  13                  1313            B      I G I I. 19
   17                  14                  4843            L     AR.R4
   17                  15                  F8A7            S     AR.MDL
   18                  16                  1213            B      I G. 19
   20                  17                  0443            L     R4.MDL
   21                  18                  0593            L     R5.MDH
   22                  19                  5801            LI    AE."01"
   22                  20                  C777            A     R7.R7
   23                  21                  4873            L     AR.R7
   23                  22                  F867            S     AR.R6
   24                  23                  1104            B      I I. 4
   25                  24                  2000            T     R0.R0
   25                  25                  1100            B      I I. 0
```

Fig 3 (b) INTERDATA MOD3 Microcode for Greatest Element Test Program

```
GREATEST_ELEMENT:PROCEDURE;
      DECLARE (I,N,GRTST_ELMNT,INDEX) BYTE;
      DECLARE ARRAY(10) BYTE;
      N=10;
      GRTST_ELMNT=ARRAY(1);
      INDEX=1;
      DO I=2 TO N;
             IF GRTST_ELMNT > ARRAY(I) THEN GO TO LAB1;
                     GRTST_ELMNT=ARRAY(I);
                     INDEX=I;
       LAB1:END;
      END;
EOF
```

Fig 4 (a) PLM Source Code for Greatest Element Test Program

```
                    QUADS GENERATED

OPERATOR  OPERAND1/   OPERAND2  RESULT/     ****  RCD_NR  LOC_QUAD
          CONDITION             LABEL
  LAB        GREATEST_ELEMENT0  0                   0        1
  ASGN       10          0      N                   0        5
  SUBS       ARRAY       1      .T3                 0        9
  ASGN       .T3         0      GRTST_ELMNT         0       13
  ASGN       1           0      INDEX               0       17
  ASGN       2           0      I                   0       21
  LAB        .L7         0      0                   0       25
  GT         I           N      .T8                 0       29
  BT         .T8         0      .L9                 0       33
  SUBS       ARRAY       I      .T10                0       37
  GT         GRTST_ELMNT .T10   .T11                0       41
  BT         .T11        0      LAB1                0       45
  SUBS       ARRAY       I      .T13                0       49
  ASGN       .T13        0      GRTST_ELMNT         0       53
  ASGN       I           0      INDEX               0       57
  LAB        LAB1        0      0                   0       61
  ADD        I           1      .T17                0       65
  ASGN       .T17        0      I                   0       69
  BR         0           0      .L7                 0       73
  LAB        .L9         0      0                   0       77
```

Fig 4 (b) Quadruple Representation of Greatest Element Test Program

Fig 4 (c) DEC PDP 11 Assembly Language Representation of Greatest Element Test Program

```
          .ENTRY    MAIN                 ;
          .WORD      0                  ;      I
          .WORD      0                  ;      N
          .WORD      0                  ;      GRTST_ELMNT
          .WORD      0                  ;      INDEX
          .WORD     0                    ;      ARRAY(1)
          .WORD     0                    ;      ARRAY(2)
          .WORD     0                    ;      ARRAY(3)
          .WORD     0                    ;      ARRAY(4)
          .WORD     0                    ;      ARRAY(5)
          .WORD     0                    ;      ARRAY(6)
          .WORD     0                    ;      ARRAY(7)
          .WORD     0                    ;      ARRAY(8)
          .WORD     0                    ;      ARRAY(9)
          .WORD     0                    ;      ARRAY(10)
          CLR       R4                   ;      MULTIPLY
          TST       R1
          BGE       .+6.
          INC       R4
          NEG       R1
          TST       R2
          BGE       .+6.
          INC       R4
          NEG       R2
          CLR       R0
          MOV       #-16.,R3
          ASL       R0
          ROL       R1
          BCC       .+6.
          ADD       R2,R0
          ADC       R1
          INC       R3
          BNE       .-12.
          TST       R1
          BNE       .+10.
          DEC       R4
          BNE       .+4
          NEG       R0
          RTS       R7
          MOV       #2.,R6
          HALT
          CLR       R0                   ;      DIVIDE
          CMP       R2,R1
          BGT       .+8.
          SUB       R2,R1
          INC       R0
          BR        .-8.
          RTS       R7
MAIN:     MOV       #4096.,R6            ;
          .WORD      6                 ;
          MOV       LABELS,R5            ;
          MOV       #10.,2.              ;
          MOV       #1.,-(R6)            ;
          ASL       (R6)                 ;
          ADD       #6.,(R6)             ;
          MOV       @(R6)+,4.            ;
          MOV       #1.,6.               ;
          MOV       #2.,0.               ;
          CMP       0.,2.                ;
          BLE       .+6                  ;
          JMP       @6.(R5)              ;
          MOV       0.,-(R6)             ;
          ASL       (R6)                 ;
          ADD       #6.,(R6)             ;
          CMP       4.,@(R6)+            ;
          BLE       .+6                  ;
          JMP       @4.(R5)              ;
          MOV       0.,-(R6)             ;
          ASL       (R6)                 ;
          ADD       #6.,(R6)             ;
          MOV       @(R6)+,4.            ;
          MOV       0.,6.                ;
          MOV       0.,-(R6)             ;
          ADD       #1.,(R6)             ;
          MOV       (R6)+,0.             ;
          JMP       @2.(R5)              ;
          HALT                           ;
LABELS:   .WORD     .+2                  ;
          .WORD     108.                 ;      GREATEST_ELEMENT
          .WORD     140.                 ;      .L7
          .WORD     192.                 ;      LAB1
          .WORD     208.                 ;      .L9
END:      .END      MAIN                 ;
$END
```

Fig 4 (c) DEC PDP 11 Assembly Language Representation of Greatest Element Test Program

```
          .ENTRY    MAIN               ;
          .WORD      0                 ;    I
          .WORD      0                 ;    N
          .WORD      0                 ;    GRTST_ELMNT
          .WORD      0                 ;    INDEX
          .WORD     0                   ;    ARRAY(1)
          .WORD     0                   ;    ARRAY(2)
          .WORD     0                   ;    ARRAY(3)
          .WORD     0                   ;    ARRAY(4)
          .WORD     0                   ;    ARRAY(5)
          .WORD     0                   ;    ARRAY(6)
          .WORD     0                   ;    ARRAY(7)
          .WORD     0                   ;    ARRAY(8)
          .WORD     0                   ;    ARRAY(9)
          .WORD     0                   ;    ARRAY(10)
          CLR       R4                  ;    MULTIPLY
          TST       R1
          BGE       .+6.
          INC       R4
          NEG       R1
          TST       R2
          BGE       .+6.
          INC       R4
          NEG       R2
          CLR       R0
          MOV       #-16.,R3
          ASL       R0
          ROL       R1
          BCC       .+6.
          ADD       R2,R0
          ADC       R1
          INC       R3
          BNE       .-12.
          TST        R1
          BNE       .+10.
          DEC       R4
          BNE       .+4
          NEG       R0
          RTS       R7
          MOV       #2.,R6
          HALT
          CLR       R0                  ;     DIVIDE
          CMP       R2,R1
          BGT       .+8.
          SUB       R2,R1
          INC       R0
          BR        .-8.
          RTS       R7
MAIN:     MOV       #4096.,R6           ;
           .WORD      6               ;
          MOV       LABELS,R5           ;
          MOV       #10.,2.             ;
          MOV       #1.,-(R6)           ;
          ASL       (R6)                ;
          ADD       #6.,(R6)            ;
          MOV       @(R6)+,4.           ;
          MOV       #1.,6.              ;
          MOV       #2.,0.              ;
          CMP       0.,2.               ;
          BLE       .+6                 ;
          JMP       @6.(R5)             ;
          MOV       0.,-(R6)            ;
          ASL       (R6)                ;
          ADD       #6.,(R6)            ;
          CMP       4.,@(R6)+           ;
          BLE       .+6                 ;
          JMP       @4.(R5)             ;
          MOV       0.,-(R6)            ;
          ASL       (R6)                ;
          ADD       #6.,(R6)            ;
          MOV       @(R6)+,4.           ;
          MOV       0.,6.               ;
          MOV       0.,-(R6)            ;
          ADD       #1.,(R6)            ;
          MOV       (R6)+,0.            ;
          JMP       @2.(R5)             ;
          HALT                          ;
LABELS:   .WORD      .+2                 ;
          .WORD     108.                ;    GREATEST_ELEMENT
          .WORD     140.                ;    .L7
          .WORD     192.                ;    LAB1
          .WORD     208.                ;    .L9
END:      .END      MAIN                ;
$END
```

TABLE II

LIST OF TEST PROGRAMS
USED TO EVALUATE THE
QUAD COMPILER

1. Fibonacci Series: Evaluates successive terms of this series.
2. Greatest Element: Finds the largest number in a list.
3. Bubble Sort: Sorts a list of numbers into ascending order.
4. Prime: Uses Sieve of Erasthenes to identify the prime numbers between 1 and N.
5. Filter: A digital low pass filter algorithm.

The next task was to select the computer   to serve as the host machine for the microcode to be generated by the quad compiler. The study contract specified the DEC VAX 11/780 system to be the host machine but we were unable to get an adequate hardware description of this system. As a result we were required to look for an alternate system. Our choice was narrowed down to a DEC machine with a horizontal control word format as similar as possible to the VAX machine (96 bit horizontal control word). Due to the availability of adequate hardware information for the DEC PDP 11/45 system, which has a 56 bit horizontal control word format with 18 control fields, this system was chosen to be the host machine.

Another reason for choice of a DEC PDP 11 computer model was the availability of DEC PDP 11 assembly language output from the quad computer. This factor facilitated the compiler evaluation activity.

A number of difficulties immediately surfaced. The DEC PDP 11/45 has no means for entering literal data at the microprogram level. This means that all addresses, indices, constants, etc. specified for program written in PLM had to be stored in main memory in the DEC PDP 11/45. There are only six registers available to the   programmer in this machine and when the number of indices, constants, and other currently processed variables exceeds six then as a backup they must be resident in main storage. At any particular point in the execution of the test algorithm only currently referenced data items are stored in the internal registers. These limitations lead to the generalized approach to handling internal register contents to be described below.

An interesting side affect of the absence of a means for introducing literal data at the microprogram level was the discovery that it was not feasible to directly microcode algorithms for the DEC PDP 11/45 when the number of constants and indices exceeded the available internal register supply. For the test cases utilized in the study, all but one (the generation of Fibonacci series values) was not directly microprogrammable in the DEC PDP 11/45. This had the undesirable side effect of eliminating one boundary of direct or hand produced microcode as a yard stick against which to measure the efficiency of the quad compiler for producing DEC PDP 11/45 microcode.

Section 4. MICROCODE GENERATION

4.1 R QUAD GENERATOR

Before describing the R Quad Generator it will be necessary to briefly describe the DEC PDP 11/45 which was selected as the host machine for the microcode to be compiled from the PLM HLL. This is a typical minicomputer architecture which was designed to efficiently interpret the DEC PDP 11 instruction set (9). An overview block diagram of this system is shown in figure 5.

As noted above, the control word for the DEC PDP 11/45 contains 56 bits and has 18 control fields. A list of these control fields and their purpose is shown in table III. Note that there are 16 registers controlled through the microprogram of which six are available for general use. Memory read access is via a BA register which transmits an address over the DEC UNIBUS to memory which returns a value one access time later over the same bus. Memory write is similar except a data value and address are transmitted to memory via the UNIBUS.

Because of the limited number of free registers in the DEC PDP 11/45 CPU, it was decided to maintain all program constants, addresses, indices and variables defined in the quadruples output by the compiler in main storage and only

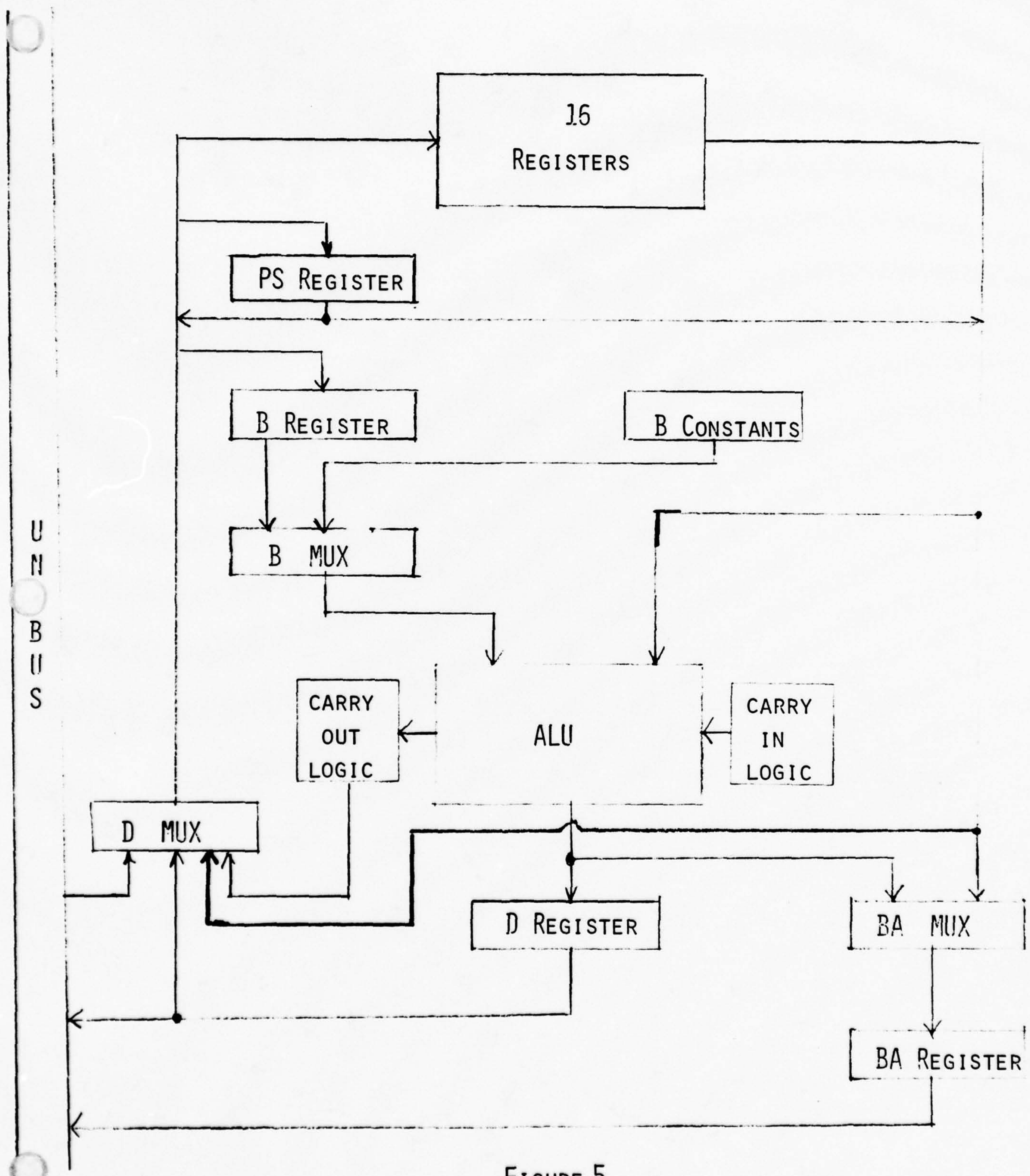


FIGURE 5

SIMPLIFIED DEC PDP 11/45 CPU BLOCK DIAGRAM

TABLE 111

DEC PDP 11/45
MICROCODE CONTROL FIELDS

1.	CLK	(3 bits)	-	Clock Control Field
2.	CIR	(1 bit)	-	Clock UNIBUS into IR Register
3.	WR	(2 bits)	-	Controls Write DMUX input to the General Registers
4.	CB	(1 bit)	-	Clock DMUX into B Register
5.	CD	(1 bit)	-	Clocks ALU Output into D Register
6.	CBA	(1 bit)	-	Clocks Data into BUS Register
7.	BUS	(3 bits)	-	BUS Control
8.	DAD	(4 bits)	-	Data Path Alteration Control
9.	SPS	(3 bits)	-	Controls Loading and Clocking of PSW Register
10.	ALU	(5 bits)	-	ALU Mode and Operation Select
11.	SBC	(4 bits)	-	Microprogram Constant Selection
12.	SBM	(4 bits)	-	BMUX Input Control
13.	SDM	(2 bits)	-	DMUX Input Control
14.	SBA	(1 bit)	-	Selects Input to BA MUX
15.	UBF	(5 bits)	-	Micro Branch Field
16.	SRX	(4 bits)	-	General Register Address Source Selection
17.	RIF	(4 bits)	-	General Register Address
18.	UPF	(8 bits)	-	Next microinstruction address

keep a limited number in registers as required at each point in the calculation. This requires keeping track of where all variables, etc. are stored in memory and which are in the registers. This is accomplished at compile time by generating a register content table which provides the required address pointers, usage, status, and register content of program data in current use. It is important to note that generation of this table doesn't impact run time execution of the microprogram but only the compilation activity.

An overview of how R quads are generated is shown at figure 6. We start with a set of quadruples generated by the compiler in which all operands are described in terms of the variables specified by the programmer of the algorithm in PLM and those variables, generated by the compiler, e.g. the indices generated to implement a DO loop or array structure. As each quadruple is read in by the R Quad generator it examines each operator and operand. The operands representing constants, indices and variables are assigned memory locations. A set of allocate and deallocate routines load and unload the program data into and out of the internal registers using read and write quadruples defined for this purpose. Operands with the same value, e.g. constants, are assigned to the same register. Variables assigned an initial value, which is already stored in a register, will be assigned to the same register. When the supply of internal registers is exhausted, an algorithm selects operands currently stored in the registers to be loaded into storage to make room for currently needed variables. As noted above, a record of all these operand assignments is being maintained during the R Quad generation process.

Two special cases must now be examined. The first is the quadruple representing a label. This quadruple is generated by the compiler in implementing a DO loop and other program structures which must serve as a point to be branched to from other parts of the microprogram. Of course, statement labels can be inserted in the HLL source as required. The second case involves

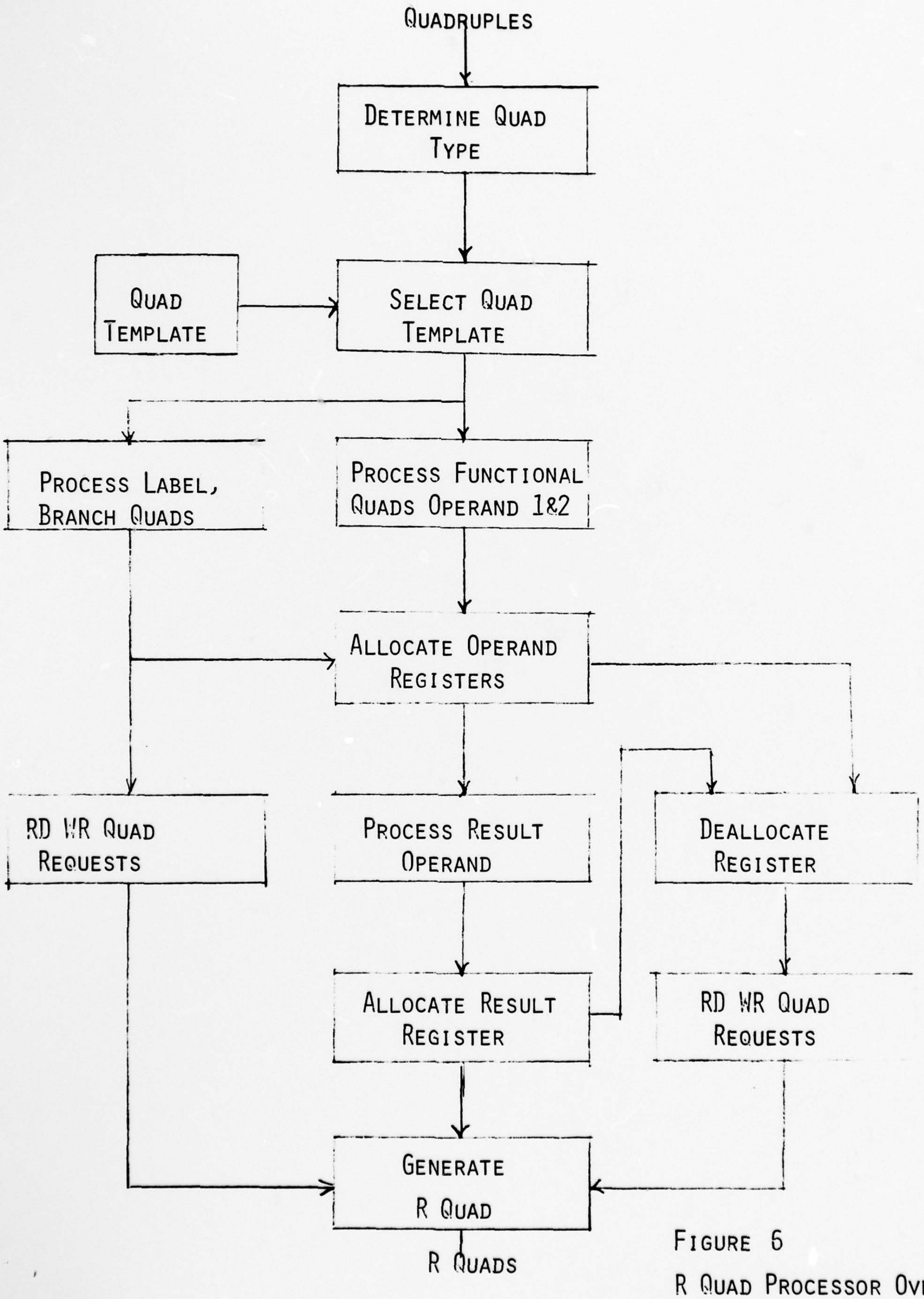


FIGURE 6

R QUAD PROCESSOR OVERVIEW

quadruples which specify branches. These may be either compiler generated or be part of the HLL source statements. In both of these cases the contents of the internal CPU registers are subject to change and the existing contents must be saved. In the case of a label, a branch may be made to this quadruple from some other quadruple in the microprogram. The register contents specified by the quadruples before the label quadruple will need to be replaced if a branch occurs to this label. In the event the program returns to this point again the old register contents must be saved. Likewise in the case of a branch quadruple it isn't known beforehand which way the branch will go. Because of the possibility of not taking the branch, the current register values are saved.

The R Quad processor assigns to registers all operands in quadruples generated by the compiler and generates the appropriate additional quadruples to move operand data between the registers and memory as required to meet this objective. The result is a set of quadruples with most operands specified in terms of internal registers. This is illustrated in figure 7, which shows the R Quads derived from the Quads representing the Greatest Element Test Program shown in Figure 4(c). A list of R Quad types is shown in table IV. The layout of the register content table required to support its operation is shown in figure 8. A detailed description of the function of each procedure of the R Quad Processor is given in appendix 7.5(a) and the R Quad generators in appendix 7.5(b).

4.2 MICROCODE GENERATOR

Each quadruple type has an operation field which designates some action to be taken by the hardware. The operands required for this action are specified and a location to store the result is given. The microcode generation routine

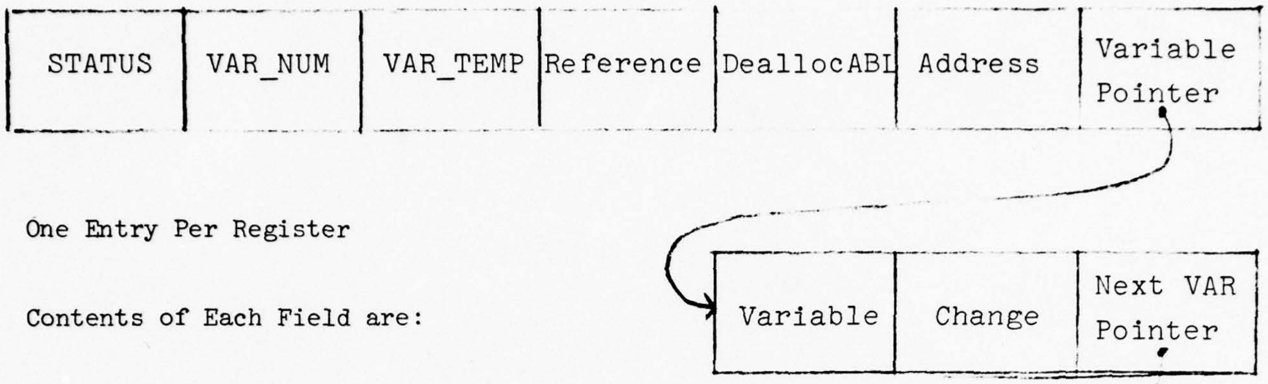
```
OPERATOR  OPERAND1/  OPERAND2  RESULT
          CONDITION            LABEL
  LAB       GREATEST_ELEMENT0    0
  RD        10         0         R6
  RDAD      ARRAY      0         R5
  RD        1          0         R4
  ASL       R4         0         R1
  ADD       R5         R1        R1
  RDVR      R1         0         R1
  RD        2          0         R3
  WT        R1         0         GRTST_ELMNT
  WT        R3         0         I
  WT        R4         0         INDEX
  WT        R6         0         N
  LAB       .L7        0         0
  RD        I          0         R6
  RD        N          0         R5
  GT        R6         R5        R1
  BT        R1         0         .L9
  RDAD      ARRAY      0         R4
  ASL       R6         0         R1
  ADD       R4         R1        R1
  RDVR      R1         0         R1
  RD        GRTST_ELMNT0         R3
  GT        R3         R1        R2
  BT        R2         0         LAB1
  ASL       R6         0         R1
  ADD       R4         R1        R1
  RDVR      R1         0         R1
  WT        R1         0         GRTST_ELMNT
  WT        R6         0         INDEX
  LAB       LAB1       0         0
  RD        I          0         R6
  RD        1          0         R5
  ADD       R6         R5        R1
  WT        R1         0         I
  BR        0          0         .L7
  LAB       .L9        0         0
```

Figure 7 R Quad Representation of Greatest Element Test Program derived from Quads Shown in Figure 4(c)

TABLE IV

REGISTER QUADS WITH OPERANDS REPRESENTING INTERNAL REGISTERS OR ADDRESSES

OPERATION	OPERAND 1	OPERAND 2	RESULT	ACTION
RD	Addr	-	Reg n	Read Content of Addr into Register n
WT	Reg n	-	Addr	Write Register n into addr.
RDAD	Var	-	Reg n	Read Addr of Varaible into Register n
RDVR	Reg n	-	Reg m	Read value in Reg m from address specified in Reg n
WTAD	Reg n	-	Reg m	Write contents of Reg n into addr in Reg m
ASL	Reg n	-	Reg m	Write contents of Reg n into Reg m shifted left one place



One Entry Per Register

One Entry per Variable Assigned to this Register

Contents of Each Field are:

Status: it indicates its associated register is allocated to a variable or not, it is a two valued entry 'FREE' or 'ALLOCATED'

VAR_NUM: it counts number of variables assigned to the register

VAR_TEMP: it counts number of temporary variables assigned to the register

Reference: its contents is a number to indicate the last reference to the register relative to the reference to the other registers.

Deallocabl: it indicates if the register can be used in the quad under process or not, its value is 'YES' or 'NO'

Address: it is a two valued element to indicate if the contents of the register is value of or address of a variable, it can take value of 'YES' or 'NO'

Var_Pointer: a pointer to list of variables assigned to the register

Variable: variable name assigned to the register

Change: this entry indicates if the variable has changed its value after the last time that it has been read from the memory

Next_Var_PTR: it is a pointer to the next variable associated to the same register

FIGURE 8

R QUAD PROCESSOR REGISTER CONTENT TABLE FORMAT consists of a case statement which causes a branch to a routine for each quadruple type. This routine refers to a tabulation of the actions carried out by each control field of the DEC PDP 11/45 and selects the appropriate binary value to actuate the required function. More than one microinstruction may be generated for a given quadruple. In figure 9 the microprogram corresponding to the R Quads shown in figure 7 is shown.

To actually use the microprogram in the DEC PDP 11/45 it would be necessary to created a load module. This would require creation of a control storage address table which provides relative address values for all operands specifying addresses. This process would be similar to that required for a conventional loader routine used to create load modules for machine language programs.

Section 5.0 PERFORMANCE EVALUATION

5.1 EVALUATION APPROACH

One of the questions concerning use of a compiler to generate microprograms is the ability to produce "efficient" microcode. The meaning of the word "efficient" is somewhat arbitrary. To develop a precise meaning is cumbersome, but in general it means the generation of micrograms which run in a minimum time. As noted above, two measures which have been adopted for expressing "efficiency" for this study, are the number of microinstructions generated to represent the algorithm and the corresponding number of main memory references. Since we were unable to execute the microprograms generated in this study, the more meaningful criteria of execution time wasn't obtainable.

As noted above, a comparison was made of alternate ways of generating DEC PDP 11/45 microcode. One was to produce DEC PDP 11/45 microcode directly using the Quad Compiler. Another way was to take the DEC PDP 11/45 assembly language statements produced by the Quad Compiler and convert these into

GENERATED MICRO WORDS

LOC	CLK	CIR	WR	CB	CD	CBA	BUS	DAD	SPS	ALU	SBC	SBM	SDM	SBA	UBF	SRX	RIF	UPF
000	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	00000001
00[illegible]	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	00000010
00[illegible]	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00000011
003	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00000100
004	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00000101
005	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0110	00000110
006	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00000111
007	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0101	00001000
010	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00001001
011	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00001010
012	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00001011
013	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0100	00001100
014	100	0	00	0	1	0	000	0000	000	01100	0000	0000	00	0	00000	0001	0100	00001101
015	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00001110
016	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0001	00001111
017	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0101	00010000
020	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00010001
021	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0001	00010010
022	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0001	00010011
023	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00010100
024	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00010101
025	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00010110
026	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0011	00010111
027	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00011000
030	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00011001
031	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	00011010
032	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0001	00011011
033	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00011100
034	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00011101
035	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	00011110
036	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0011	00011111
037	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00100000
040	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00100001
041	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	00100010
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	000	00000	0000	0000	00	0	00000	0001	0100	00100011
043	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00100100
044	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00100101
045	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	00100110
046	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0110	00100111
047	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	00101000
050	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	00101001
051	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00101010
052	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00101011
053	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00101100
054	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0110	00101101
055	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00101110
056	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00101111
057	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00110000
060	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0101	00110001
061	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0101	00110010
062	100	0	00	0	1	0	000	1000	011	00110	0000	0000	00	0	00000	0001	0110	00110011
063	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00110100
064	100	0	00	0	1	0	000	0000	001	10011	0001	1111	00	0	01010	0000	0000	00110101
065	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	00110110
066	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	00111000
067	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	01101100
070	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00111001
071	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0100	00111010
072	100	0	00	0	1	0	000	0000	000	01100	0000	0000	00	0	00000	0001	0110	00111011
073	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00111100
074	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0001	00111101
075	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0100	00111110
076	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00111111
077	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0001	01000000
100	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0001	01000001
101	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	01000010
102	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	01000011
103	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	01000100
104	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0011	01000101
105	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0001	01000110
106	100	0	00	0	1	0	000	1000	011	00110	0000	0000	00	0	00000	0001	0011	01000111
107	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0010	01001000
110	100	0	00	0	1	0	000	0000	001	10011	0001	1111	00	0	01010	0000	0000	01001001
111	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	01001010
112	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	01001100

DF.

FIGURE 9 DEC PDP 11/45 Microcode Generated by PLM to Quad Compiler for Greatest Elements Test Program

```
  3  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 01011011
  4  100 0 00 0 1 0 000 0000 000 01100 0000 0000 00 0 00000 0001 0110 01001101
115  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 01001110
116  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0001 01001111
117  100 0 00 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0100 01010000
120  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 01010001
121  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0001 01010010
122  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0001 01010011
123  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01010100
124  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 01010101
125  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 1000 01010110
126  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0001 01010111
127  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01011000
130  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 01011001
131  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 1000 01011010
132  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0110 01011011
133  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0000 01011100
134  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0000 01011101
135  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01011110
136  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 01011111
137  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 01100000
140  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0110 01100001
141  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01100010
142  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 01100011
143  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 01100100
144  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0101 01100101
145  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0101 01100110
146  100 0 00 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0110 01100111
147  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 01101000
150  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01101001
151  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 01101010
152  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 1000 01101011
153  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0001 00100111
154  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0000 01101101
1    010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0000 01101110
```

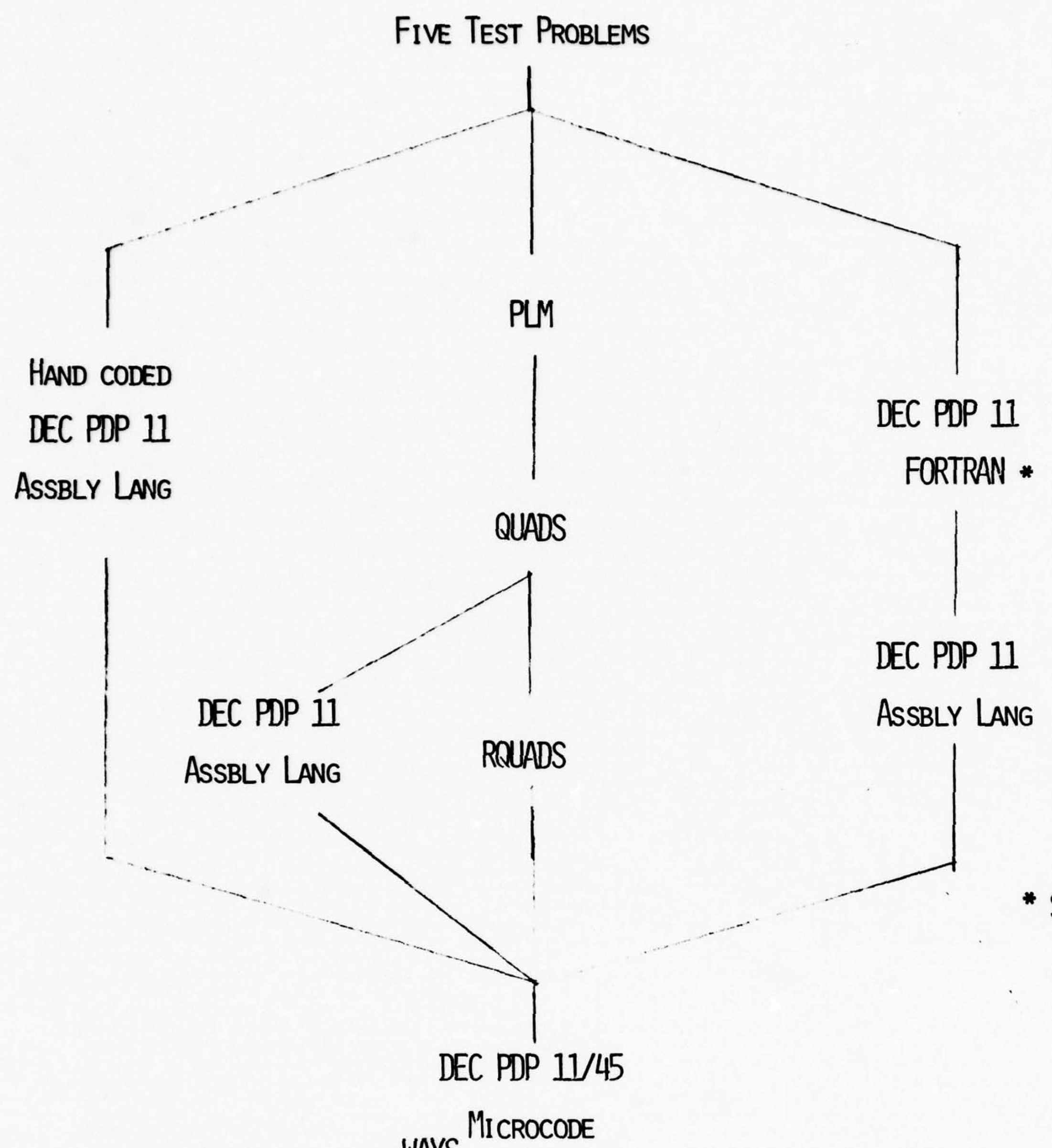
FIGURE 9 Continued

equivalent microcode. A third way was to hand-code a DEC PDP 11/45 assembly language version and then convert that into equivalent microcode. The final way is to use a DEC supplied FORTRAN Compiler which produces DEC PDP 11 assembly language statments which are then converted into microcode. Figure 10 illustrates these alternate paths for generating microcode and the comparative results are shown in Table V.

As noted above, only the Fibonacci Series test problem could be directly hand coded into microcode. The number of microinstructions for this case was 11 compared to 83 generated by the Quad Compiler. It should be noted that for this test case all indices and variables could be left in internal DEC PDP 11/45 registers and no memory references were required. This shows that hand-coded microcode can be very efficient but of very limited application.

5.2 QUAD COMPILER PERFORMANCE

The data shown in Table V indicates that for the very simple test cases (Fibonacci Series, Greatest-Element) the hand-coded DEC PDP 11 assembly langauge implementation is more efficient than the quad compiler implementation. For the more complex test cases (digital filter and Prime Number Generator) the quad compiler is more efficient. Another interesting comparison is to the DEC PDP 11 machine language statments produced by the quad compiler from the quad representation. It has been well established (10) that assembly language implementations produced by compilers aren't as efficient as hand coded assembly language, which is indicated by the results shown. Going directly from quads to microcode appears to be about 4 times more efficient than going from quads to DEC PDP 11 machine language and then to microcode for the more complicated test cases. The DEC PDP 11 assembly language to microcode alternative produces from 1.5 to 2 times as many microinstructions as the direct PLM to microcode via



* See note on pg. 29

Figure 10 ALTERNATE WAYS TO GENERATE DEC PDP 11/45 MICROCODE

TABLE V

COMPARISON OF ALTERNATE METHODS FOR GENERATION OF MICROCODE FOR THE DEC PDP 11/45

	Compiler Generated Quads	Compiler Generated R Quads			Compiler Generated Assembly Language			Hand Coded Assembly Language		
	Number	Number	Micro Instruction	Memory Reference	Number	Micro Instruction	Memory Reference	Number	Micro Instruction	Memory Reference
Fibonacci Series	16	24	83	32	17	163	54	12	50	22
Greatest Element	20	36	109	35	26	231	91	14	77	29
Sort	31	56	165	50	45	399	171	21	141	55
Prime Number	35	57	182	50	44	473	181	26	235	72
Digital Filter	28	53	161	38	51	574	214	24	276	74

* See note Pg. 29

compiler method. In general the PLM to microcode compiler appears to be comparable with hand coded assembly language in spite of the necessity to move register data in and out of main storage for label and branch quadruples. To date no attempt has been made to optimize the PLM to quad compiler or the R Quad generation process. It is our intention to pursue these alternatives as ongoing research.

As indicated by the asterisks on figure 10 and table V, we had intended to compare the performance of the PLM to DEC PDP 11/45 compiler with a standard commercially available FORTRAN compiler supplied for the DEC PDP 11 systems. FORTRAN versions of the five test programs have been prepared but we are unable to get a machine language listing (object code) with any of the compilers we have access to. Accordingly, the corresponding efficiency comparison to the DEC FORTRAN compiler isn't shown in table V. We anticipate obtaining this data in the near future and it will be supplied as an addendum to this report.

SECTION 6.0 RECOMMENDATIONS FOR FURTHER RESEARCH

6.1 Conversion of Input Language of Quad Compiler to PASCAL

PASCAL has been widely accepted as a high level language used for scientific research. The U.S. Army BMD DDP Test Bed has selected PASCAL as its standard language. Accordingly, it is desirable to have this language as an input to a compiler producing microcode for the DEC VAX 11/780. PASCAL has a considerably more complex syntax structure than PLM and some problems will have to be resolved to convert the present quad compiler to accept PASCAL as an input language. It is very likely that due to limitations in the XPL

TWS, we may not be able to implement the entire PASCAL language but only a subset. The selection of this subset is a research issue. A proposal to carry out this research has been accepted and is about to begin.

6.2 Optimization of Quad Compiler

There has been no attempt made to optimize the performance of the quad compiler. There are two levels of optimization that can be attempted. The first would be to optimize the generation of quadruples by the complier and the second would be to optimize the generation of R Quads. It is felt that taking advantage of both of these optimization prospects should greatly enhance the performance of the PASCAL to Quad to DEC VAX 11/780 compiler. Since it is necessary to get the compiler converted to the PASCAL input language producing microcode for the DEC VAX 11/780, research into introducing optimization into this compiler must be deferred. A proposal indicating several approaches to quad compiler optimization will be generated in the near future.

6.3 Microprogram Host Machine Interface Study

In developing a compiler to produce microcode for a host machine with a horizontally encoded control word, it quickly became apparent that the host machine architecture had a profound impact on the ease with which the compiling process could be carried out. As noted above, the DEC PDP 11/45 has serious limitations for hand coded microprograms and it became apparent that there was a preferred host machine architecture if an intermediate program representation was going to be used, i.e., quadruple or triple format. Tuning of the architecture to optimize performance of compiled microcode via an intermediate representation could also lead to a less complex compiler internal structure. A study of these tradeoffs among host machine architecture, intermediate program representation, and the complexity of the microcode compiler would have great research value. As more understanding of the host machine architecture and microcode interface is gained as a result of designing the PASCAL to DEC VAX 11/780 microcode compiler, a detailed research study proposal will be prepared to explore the issues noted above in more detail.

REFERENCES

1. Eckhouse, P.H., "A High Level Microprogramming Language (MPL)," PH.D. Dissertation, State University of New York at Buffalo, 1971.

2. Fodor, P.R., "Evaluation of Compiler Design and Performance for a Virtual Microprogrammed Machine," M.S. Dissertation, The George Washington University, 1976.

3. Abd-Alla, A.M. and Karlgaard, D.C., "Heuristic Synthesis of Microprogrammed Computer Architecture," IEEE TC, Vol. C-23, pp. 802-808, August, 1974.

4. Moffett, L.H., "On-line Virtual Architecture Timing Using Micro-capture," Ph.D. Dissertation, The George Washington University, 1976.

5. Evans, R.H., "Architecture Considerations in Signal Sorter Design," Ph.D. Dissertation, The George Washington University, 1976.

6. McKeeman, W.M., Horning, J.J. and Wortman, D.B., A Compiler Generator, Prentice-Hall, 1970.

7. Kildall, G.A., "High Level Language Simplifies Microcomputer Programming," Electronics, June, 1974.

8. Gries, David, "Compiler Construction for Digital Computers," John Wiley 1971.

9. Digital Equip. Corp., "Processor Handbook PDP 11," Digital Equip. Corp. 1975.

10. Henriksen, J.O., Merwin, R.E., "Programming Language Efficiency in Real-Time Software Systems," AFIPS Proc. SJCC, 40 (1972) pp. 155-161.

APPENDIX 7.1 - 1

PLM TO DEC PDP 11/45   MICROCODE COMPILER LISTING

```
| /*   SKELETON                                                                        | 1286
|                  THE PROTO-COMPILER OF THE XPL SYSTEM                                | 1286
|                                                                                      | 1286
|                                                                                      | 1286
|W. M. MCKEEMAN           J. J. HORNING           D. B. WORTMAN                        | 1286
|                                                                                      | 1286
|INFORMATION &            COMPUTER SCIENCE        COMPUTER SCIENCE                     | 1286
|COMPUTER SCIENCE,        DEPARTMENT,             DEPARTMENT,                          | 1286
|                                                                                      | 1286
|UNIVERSITY OF            STANFORD                STANFORD                             | 1286
|CALIFORNIA AT            UNIVERSITY,             UNIVERSITY,                          | 1286
|                                                                                      | 1286
|SANTA CRUZ,              STANFORD,               STANFORD,                            | 1286
|CALIFORNIA               CALIFORNIA              CALIFORNIA                           | 1286
|95060                    94305                   94305                                | 1286
|                                                                                      | 1286
|DEVELOPED AT THE STANFORD COMPUTATION CENTER, CAMPUS FACILITY,   1966-69              | 1286
|AND THE UNIVERSITY OF CALIFORNIA COMPUTATION CENTER, SANTA CRUZ, 1968-69.             | 1286
|                                                                                      | 1286
|DISTRIBUTED THROUGH THE SHARE ORGANIZATION.                                           | 1286
|THIS VERSION OF SKELETON IS A SYNTAX CHECKER FOR THE FOLLOWING GRAMMAR:               | 1286
| /*  <PROGRAM> ::= <STMT LIST> EOF     */                                             | 1286
| /*  <STMT LIST> ::= <STMT>     */                                                    | 1286
| /*  <STMT LIST> ::= <STMT LIST> <STMT>     */                                        | 1286
| /*  <STMT> ::= <BASIC STMT>     */                                                   | 1286
| /*  <STMT> ::= <IF STMT>     */                                                      | 1286
| /*  <BASIC STMT> ::= <ASSIGNMENT> ;     */                                           | 1286
| /*  <BASIC STMT> ::= <GROUP> ;     */                                                | 1286
| /*  <BASIC STMT> ::= <PROC DEF> ;     */                                             | 1286
| /*  <BASIC STMT> ::= <RETURN STMT> ;     */                                          | 1286
| /*  <BASIC STMT> ::= <CALL STMT> ;     */                                            | 1286
| /*  <BASIC STMT> ::= <GO TO STMT> ;     */                                           | 1286
| /*  <BASIC STMT> ::= <DECLARATION STMT> ;     */                                     | 1286
| /*  <BASIC STMT> ::= ;     */                                                        | 1286
| /*  <BASIC STMT> ::= <LABEL DEF> <BASIC STMT>     */                                 | 1286
| /*  <IF STMT> ::= <IF CLAUSE> <STMT>     */                                          | 1286
| /*  <IF STMT> ::= <IF CLAUSE> <TRUE PART> <STMT>     */                              | 1286
| /*  <IF STMT> ::= <LABEL DEF> <IF STMT>     */                                       | 1286
| /*  <IF CLAUSE> ::= IF <EXPR> THEN     */                                            | 1286
| /*  <TRUE PART> ::= <BASIC STMT> ELSE     */                                         | 1286
| /*  <GROUP> ::= <GROUP HEAD> <ENDING>     */                                         | 1286
| /*  <GROUP HEAD> ::= DO ;     */                                                     | 1286
| /*  <GROUP HEAD> ::= DO <STEP DEF> ;     */                                          | 1286
| /*  <GROUP HEAD> ::= DO <WHILE CLAUSE> ;     */                                      | 1286
| /*  <GROUP HEAD> ::= DO <CASE SELECTOR> ;     */                                     | 1286
| /*  <GROUP HEAD> ::= <GROUP HEAD> <STMT>     */                                      | 1286
| /*  <STEP DEF> ::= <VAR> <REPLACE> <EXPR> <ITERATION CONTROL>     */                 | 1286
| /*  <ITERATION CONTROL> ::= TO <EXPR>     */                                         | 1286
| /*  <ITERATION CONTROL> ::= TO <EXPR> BY <EXPR>     */                               | 1286
| /*  <WHILE CLAUSE> ::= WHILE <EXPR>     */                                           | 1286
| /*  <CASE SELECTOR> ::= CASE <EXPR>     */                                           | 1286
| /*  <PROC DEF> ::= <PROC HEAD> <STMT LIST> <ENDING>     */                           | 1286
| /*  <PROC HEAD> ::= <PROC NAME> ;     */                                             | 1286
| /*  <PROC HEAD> ::= <PROC NAME> <TYPE> ;     */                                      | 1286
| /*  <PROC HEAD> ::= <PROC NAME> <PARAMETER LIST> ;     */                            | 1286
| /*  <PROC HEAD> ::= <PROC NAME> <PARAMETER LIST> <TYPE> ;     */                     | 1286
| /*  <PROC NAME> ::= <LABEL DEF> PROCEDURE     */                                     | 1286
| /*  <PARAMETER LIST> ::= <PARAMETER HEAD> <IDENTIFIER> )     */                      | 1286
```

```
     |  [illegible]                                                                      |  [illegible]
 60  |  /*  <PARAMETER HEAD> ::= <PARAMETER HEAD> <IDENTIFIER> ,     */                  |  1286
 61  |  /*  <ENDING> ::= END     */                                                       |  1286
 62  |  /*  <ENDING> ::= END <IDENTIFIER>     */                                          |  1286
 63  |  /*  <ENDING> ::= <LABEL DEF> <ENDING>     */                                      |  1286
 64  |  /*  <RETURN STMT> ::= RETURN     */                                               |  1286
 65  |  /*  <RETURN STMT> ::= RETURN <EXPR>     */                                        |  1286
 66  |  /*  <CALL STMT> ::= CALL <VARIABLE>     */                                        |  1286
 67  |  /*  <GO TO STMT> ::= <GO TO> <IDENTIFIER>     */                                  |  1286
 68  |  /*  <GO TO> ::= GO TO     */                                                      |  1286
 69  |  /*  <GO TO> ::= GOTO     */                                                       |  1286
 70  |  /*  <DECLARATION STMT> ::= DECLARE <DECLARATION ELEMENT>     */                   |  1286
 71  |  /*  <DECLARATION ELEMENT> ::= <IDENTIFIER> LITERALLY <STRING>     */              |  1286
 72  |  /*  <TYPE DECLARATION> ::= <IDENTIFIER SPECIFICATION> <TYPE>     */               |  1286
 73  |  /*  <DECLARATION STMT> ::= <DECLARATION STMT> , <DECLARATION ELEMENT>     */      |  1286
 74  |  /*  <DECLARATION ELEMENT> ::= <TYPE DECLARATION>     */                           |  1286
 75  |  /*  <TYPE DECLARATION> ::= <BOUND HEAD> <NUMBER> ) <TYPE>     */                  |  1286
 76  |  /*  <TYPE DECLARATION> ::= <TYPE DECLARATION> <INITIAL LIST>     */               |  1286
 77  |  /*  <TYPE> ::= STATUS     */                                                      |  1286
 78  |  /*  <TYPE> ::= REGISTER ( NUMBER )     */                                         |  1286
 79  |  /*  <TYPE> ::= MEMORY ( NUMBER )     */                                           |  1286
 80  |  /*  <BOUND HEAD> ::= <IDENTIFIER SPECIFICATION> (     */                          |  1286
 81  |  /*  <IDENTIFIER SPECIFICATION> ::= <IDENTIFIER>     */                            |  1286
 82  |  /*  <IDENTIFIER SPECIFICATION> ::= <IDENTIFIER LIST> <IDENTIFIER> )     */        |  1286
 83  |  /*  <IDENTIFIER LIST> ::= (     */                                                |  1286
 84  |  /*  <IDENTIFIER LIST> ::= <IDENTIFIER LIST> <IDENTIFIER> ,     */                 |  1286
 85  |  /*  <INITIAL LIST> ::= <INITIAL HEAD> <CONSTANT> )     */                         |  1286
 86  |  /*  <INITIAL HEAD> ::= INITIAL (     */                                           |  1286
 87  |  /*  <INITIAL HEAD> ::= <INITIAL HEAD> <CONSTANT> ,     */                         |  1286
 88  |  /*  <ASSIGNMENT> ::= <VAR> <REPLACE> <EXPR>     */                                |  1286
 89  |  /*  <ASSIGNMENT> ::= <LEFT PART> <ASSIGNMENT>     */                              |  1286
 90  |  /*  <REPLACE> ::= =     */                                                        |  1286
 91  |  /*  <LEFT PART> ::= <VARIABLE> ,     */                                           |  1286
 92  |  /*  <EXPR> ::= <LOGICAL FACTOR>     */                                            |  1286
 93  |  /*  <EXPR> ::= <EXPR> | <LOGICAL FACTOR>     */                                   |  1286
 94  |  /*  <LOGICAL FACTOR> ::= <LOGICAL SECONDARY>     */                               |  1286
 95  |  /*  <LOGICAL FACTOR> ::= <LOGICAL FACTOR> & <LOGICAL SECONDARY>     */            |  1286
 96  |  /*  <LOGICAL FACTOR> ::= <LOGICAL FACTOR> XOR <LOGICAL SECONDARY>     */          |  1286
 97  |  /*  <LOGICAL SECONDARY> ::= <LOGICAL PRIMARY>     */                              |  1286
 98  |  /*  <LOGICAL SECONDARY> ::= ¬ <LOGICAL PRIMARY>     */                            |  1286
 99  |  /*  <LOGICAL PRIMARY> ::= <STRING EXPR>     */                                    |  1286
100  |  /*  <LOGICAL PRIMARY> ::= <STRING EXPR> <RELATION> <STRING EXPR>     */           |  1286
101  |  /*  <RELATION> ::= =     */                                                       |  1286
102  |  /*  <RELATION> ::= <     */                                                       |  1286
103  |  /*  <RELATION> ::= >     */                                                       |  1286
104  |  /*  <RELATION> ::= ¬ =     */                                                     |  1286
105  |  /*  <RELATION> ::= ¬ <     */                                                     |  1286
106  |  /*  <RELATION> ::= ¬ >     */                                                     |  1286
107  |  /*  <RELATION> ::= < =     */                                                     |  1286
108  |  /*  <RELATION> ::= > =     */                                                     |  1286
109  |  /*  <STRING EXPR> ::= <ARITH EXPR>     */                                         |  1286
110  |  /*  <STRING EXPR> ::= <STRING EXPR> || <ARITH EXPR>     */                        |  1286
111  |  /*  <ARITH EXPR> ::= <TERM>     */                                                |  1286
112  |  /*  <ARITH EXPR> ::= <ARITH EXPR> + <TERM>     */                                 |  1286
113  |  /*  <ARITH EXPR> ::= <ARITH EXPR> - <TERM>     */                                 |  1286
114  |  /*  <ARITH EXPR> ::= + <TERM>     */                                              |  1286
115  |  /*  <ARITH EXPR> ::= - <TERM>     */                                              |  1286
116  |  /*  <TERM> ::= <PRIMARY>     */                                                   |  1286
117  |  /*  <PRIMARY> ::= <CONSTANT>     */                                               |  1286
118  |  /*  <PRIMARY> ::= <VAR>     */                                                    |  1286
119  |  /*  <PRIMARY> ::= ( <EXPR> )     */                                               |  1286
120  |  /*  <CONSTANT> ::= <NUMBER>     */                                                |  1286
121  |  /*  <VARIABLE> ::= <IDENTIFIER>     */                                            |  1286
122  |  /*  <VARIABLE> ::= <SUBSCRIPT HEAD> <EXPR> )     */                               |  1286
123  |  /*  <SUBSCRIPT HEAD> ::= <IDENTIFIER> (     */                                    |  1286
124  |  /*  <SUBSCRIPT HEAD> ::= <SUBSCRIPT HEAD> <EXPR> ,     */                         |  1286
```

```
[illegible]
DECLARE V(NSY) CHARACTER INITIAL ( '<ERROR: TOKEN = 0>', ';', ')', '(', ',',
   ':', '=', '<', '>', '+', '-', '*', '/', '.', 'IF', 'DO', 'GO', 'TO', ':=',
   'OR', 'BY', '_|_', 'END', 'XOR', 'AND', 'NOT', 'MOD', 'HALT', 'THEN',
   'ELSE', 'CASE', 'CALL', 'GOTO', 'DATA', 'BYTE', 'PLUS', 'LABEL', 'BASED',
   'MINUS', 'WHILE', 'ENABLE', 'RETURN', 'DISABLE', 'DECLARE', 'ADDRESS',
   'INITIAL', '<NUMBER>', '<STRING>', 'INTERRUPT', 'PROCEDURE', 'LITERALLY',
   '<IDENTIFIER>', '<TO>', '<BY>', '<TYPE>', '<COMP>', '<TERM>', '<GROUP>',
   '<WHILE>', '<GO TO>', '<ENDING>', '<PROGRAM>', '<REPLACE>', '<PRIMARY>',
   '<VARIABLE>', '<CONSTANT>', '<RELATION>', '<STATEMENT>', '<IF CLAUSE>',
   '<TRUE PART>', '<DATA LIST>', '<DATA HEAD>', '<LEFT PART>',
   '<ASSIGNMENT>', '<EXPRESSION>', '<GROUP HEAD>', '<BOUND HEAD>',
   '<IF STATEMENT>', '<WHILE CLAUSE>', '<INITIAL LIST>', '<INITIAL HEAD>',
   '<CASE SELECTOR>', '<VARIABLE NAME>', '<CONSTANT HEAD>',
   '<STATEMENT LIST>', '<CALL STATEMENT>', '<PROCEDURE HEAD>',
   '<PROCEDURE NAME>', '<PARAMETER LIST>', '<PARAMETER HEAD>',
   '<BASED VARIABLE>', '<LOGICAL FACTOR>', '<SUBSCRIPT HEAD>',
   '<BASIC STATEMENT>', '<GO TO STATEMENT>', '<STEP DEFINITION>',
   '<IDENTIFIER LIST>', '<LOGICAL PRIMARY>', '<RETURN STATEMENT>',
   '<LABEL DEFINITION>', '<TYPE DECLARATION>', '<ITERATION CONTROL>',
   '<LOGICAL SECONDARY>', '<LOGICAL EXPRESSION>', '<DECLARATION ELEMENT>',
   '<PROCEDURE DEFINITION>', '<DECLARATION STATEMENT>',
   '<ARITHMETIC EXPRESSION>', '<IDENTIFIER SPECIFICATION>');
DECLARE V_INDEX(12) BIT(8) INITIAL ( 1, 14, 21, 27, 36, 40, 42, 46, 48, 51,
   51, 51, 52);
DECLARE C1(NSY) BIT(104) INITIAL (
   "(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 10000 00",
   "(2) 02000 00000 00002 22000 02200 00202 02200 00000 22220 10000 02",
   "(2) 02222 02222 22200 00222 20022 02020 00003 23020 00003 10000 00",
   "(2) 00030 00000 30030 00000 00000 30000 00000 00000 00000 10300 03",
   "(2) 00030 00000 20020 00000 00000 20000 00000 00000 00000 12200 03",
   "(2) 02000 00000 00002 22000 02200 00200 02200 00000 22220 12002 02",
   "(2) 00020 00000 20020 00000 00000 20000 00000 00000 00000 12200 02",
   "(2) 00020 01010 20020 00000 00000 00000 00000 00000 00000 12200 02",
   "(2) 00020 01000 20020 00000 00000 00000 00000 00000 00000 12200 02",
   "(2) 00010 00000 00010 00000 00000 00000 00000 00000 00000 11100 01",
   "(2) 00010 00000 00010 00000 00000 00000 00000 00000 00000 11100 01",
   "(2) 00010 00000 00010 00000 00000 00000 00000 00000 00000 11100 01",
   "(2) 00010 00000 00010 00000 00000 00000 00000 00000 00000 11100 01",
   "(2) 00010 00000 00000 00000 00100 00000 00000 00000 00000 11100 01",
   "(2) 00010 00000 10010 00000 00000 10000 00000 00000 00000 11100 01",
   "(2) 01000 00000 00000 00000 00000 00000 10000 00001 00000 10000 01",
   "(2) 00000 00000 00000 00100 00000 00000 00000 00000 00000 10000 00",
   "(2) 00020 00000 20120 00000 00000 20000 00000 00000 00000 12200 02",
   "(2) 00010 00000 10010 00000 00000 10000 00000 00000 00000 11100 01",
   "(2) 00010 00000 10010 00000 00000 10000 00000 00000 00000 11100 01",
   "(2) 00020 00100 20020 00000 00000 20000 00000 00000 00000 12200 02",
   "(2) 01000 00000 00001 11000 00000 00100 01100 00000 11110 11000 01",
   "(2) 02000 00000 00100 00000 00000 00000 00000 00000 00000 10000 01",
   "(2) 00010 00000 10010 00000 00000 10000 00000 00000 00000 11100 01",
   "(2) 00010 00000 10010 00000 00000 10000 00000 00000 00000 11100 01",
   "(2) 00010 00000 10010 00000 00000 00000 00000 00000 00000 11100 01",
   "(2) 00010 00000 00010 00000 00000 00000 00000 00000 00000 11100 01",
   "(2) 01000 00000 00000 00000 00000 00000 00000 00000 00000 10000 00",
   "(2) 02000 00000 00002 22000 00000 00200 02200 00000 22220 12000 02",
   "(2) 02000 00000 00002 22000 00000 00200 02200 00000 22220 12000 02",
   "(2) 00010 00000 10010 00000 00000 10000 00000 00000 00000 11100 01",
   "(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 10000 01",
   "(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 10000 02",
   "(2) 00010 00000 00000 00000 00000 00000 00000 00000 00000 10000 00",
   "(2) 02002 00000 00000 00000 00000 00000 00000 00000 00000 10000 00",
   "(2) 00010 00000 00010 00000 00000 00000 00000 00000 00000 11100 01",
   "(2) 02002 00000 00000 00000 00000 00000 00000 00000 00000 10000 00",
   "(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 10000 02",
   "(2) 00010 00000 00010 00000 00000 00000 00000 00000 00000 11100 01",
   "(2) 00020 00000 20020 00000 00000 20000 00000 00000 00000 12200 02",
191 |        "(2) 01000 [illegible]                                                   ",     |  [illegible]
192 |        "(2) 02010 00000 10010 00000 00000 10000 00000 00000 00000 01100 01",     |  1286
193 |        "(2) 01000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00",     |  1286
194 |        "(2) 00010 00000 00000 00000 00000 00000 00000 00000 00000 00000 01",     |  1286
195 |        "(2) 02002 00000 00000 00000 00000 00000 00000 00000 00000 10000 00",     |  1286
196 |        "(2) 00010 00000 00000 00000 00000 00000 00000 00000 00000 00000 00",     |  1286
197 |        "(2) 03302 12222 22200 00202 20022 02020 00000 20020 00000 01000 00",     |  1286
198 |        "(2) 02202 02222 22200 00202 20022 02020 00000 20020 00000 01000 00",     |  1286
199 |        "(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 01000 00",     |  1286
200 |        "(2) 02020 00000 00000 00000 00000 00000 00002 02000 00002 10000 00",     |  1286
201 |        "(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 10100 00",     |  1286
202 |        "(2) 02333 12222 22200 00222 20022 02020 00012 22120 00002 10000 10",     |  1286
203 |        "(2) 00010 00000 10010 00000 00000 10000 00000 00000 00000 11100 01",     |  1286
204 |        "(2) 00010 00000 10010 00000 00000 10000 00000 00000 00000 01100 01",     |  1286
205 |        "(2) 03002 00000 00000 00000 00000 00000 00000 00000 00000 10000 00",     |  1286
206 |        "(2) 02020 00000 20020 00000 00000 00000 00000 00000 00000 12200 02",     |  1286
207 |        "(2) 02202 02222 21100 00202 20022 01020 00000 20020 00000 10000 00",     |  1286
208 |        "(2) 01000 00000 00000 00000 00000 00000 00000 00000 00000 10000 00",     |  1286
209 |        "(2) 00010 00000 10010 00000 00000 10000 00000 00000 00000 11100 01",     |  1286
210 |        "(2) 00000 00000 00000 00100 00000 00000 00000 00000 00000 11000 01",     |  1286
211 |        "(2) 02000 01000 00000 00000 00000 00000 00000 00000 00000 11000 00",     |  1286
212 |        "(2) 00000 01000 00000 00000 02000 00000 00000 00000 00000 10000 00",     |  1286
213 |        "(2) 00010 00000 10010 00000 00000 10000 00000 00000 00000 11100 01",     |  1286
214 |        "(2) 02202 02222 22200 00202 20022 02020 00000 20020 00000 10000 00",     |  1286
215 |        "(2) 02203 03222 22200 00212 20022 02020 00000 20020 00000 10000 00",     |  1286
216 |        "(2) 02303 02222 22200 00202 20022 02020 00000 20020 00000 10000 00",     |  1286
217 |        "(2) 00010 00000 10010 00000 00000 00000 00000 00000 00000 11100 01",     |  1286
218 |        "(2) 02000 00000 00002 22000 02200 00200 02200 00000 22220 12000 02",     |  1286
219 |        "(2) 01000 00000 00001 11000 00000 00100 01100 00000 11110 11000 01",     |  1286
220 |        "(2) 01000 00000 00001 11000 00000 00100 01100 00000 11110 11000 01",     |  1286
221 |        "(2) 02002 00000 00000 00000 00000 00000 00000 00000 00000 10000 00",     |  1286
222 |        "(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 11100 00",     |  1286
223 |        "(2) 00000 00000 00000 00000 00000 00000 00000 00010 00000 10000 01",     |  1286
224 |        "(2) 03000 00000 00000 00000 00000 00000 00000 00000 00000 10000 00",     |  1286
225 |        "(2) 02101 00000 00000 00100 10000 00010 00000 00010 00000 10000 00",     |  1286
226 |        "(2) 01000 00000 00001 11000 00100 00100 01100 00000 11110 11000 01",     |  1286
227 |        "(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 11000 00",     |  1286
228 |        "(2) 02200 00000 00002 22000 02200 00200 02200 00000 22220 12000 02",     |  1286
229 |        "(2) 01000 00000 00000 00000 00000 00000 00000 00000 00000 10000 00",     |  1286
230 |        "(2) 02002 00000 00000 00000 00000 00000 00000 00000 00000 10000 00",     |  1286
231 |        "(2) 00002 00000 00000 00000 00000 00000 00000 00000 00000 11100 00",     |  1286
232 |        "(2) 01000 00000 00000 00000 00000 00000 00000 00000 00000 11000 00",     |  1286
233 |        "(2) 00121 00000 00000 00000 00000 00000 00002 02000 00002 10000 00",     |  1286
234 |        "(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 11100 00",     |  1286
235 |        "(2) 01000 00000 00001 11000 02100 00100 01100 00000 11110 11000 01",     |  1286
236 |        "(2) 01000 00000 00000 00000 00000 00000 00000 00000 00000 10000 00",     |  1286
237 |        "(2) 01000 00000 00001 11000 00000 00100 01100 00000 11110 01000 01",     |  1286
238 |        "(2) 01010 00000 00000 00000 00000 00000 00001 01000 00001 00010 00",     |  1286
239 |        "(2) 01000 00000 00000 00000 00000 00000 00001 01000 00001 00000 00",     |  1286
240 |        "(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000 01",     |  1286
241 |        "(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000 01",     |  1286
242 |        "(2) 02202 00000 00000 00202 20021 00020 00000 00000 00000 00000 00",     |  1286
243 |        "(2) 00010 00000 10010 00000 00000 10000 00000 00000 00000 01100 01",     |  1286
244 |        "(2) 02000 00000 00002 22000 02200 00203 02200 00000 22220 02000 02",     |  1286
245 |        "(2) 01000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00",     |  1286
246 |        "(2) 01000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00",     |  1286
247 |        "(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000 01",     |  1286
248 |        "(2) 02202 00000 00000 00202 20022 00020 00000 00010 00000 00000 00",     |  1286
249 |        "(2) 01000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00",     |  1286
250 |        "(2) 01000 00000 00001 11000 00100 00100 01100 00000 11110 01001 01",     |  1286
251 |        "(2) 02002 00000 00000 00000 00000 00000 00000 00000 00000 10000 00",     |  1286
252 |        "(2) 02000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00",     |  1286
253 |        "(2) 02202 00000 00000 00202 20022 00020 00000 00000 00000 00000 00",     |  1286
254 |        "(2) 02202 00000 00000 00201 20010 00020 00000 00000 00000 00000 00",     |  1286
255 |        "(2) 02002 00000 00000 00000 00000 00000 00000 00000 00000 10000 00",     |  1286
256 |        "(2) 01100 00000 01000 00000 00100 00000 00100 00000 00000 10000 00",     |  1286
```

```
[illegible]
   "(2) 02202 01111 10000 00202 20022 00020 00000 10010 00000 1.000 00",
   "(2) 00010 00000 00000 00000 00000 00000 00001 01000 00001 0.000 00");
DECLARE NCITRIPLES LITERALLY '228';
DECLARE CITRIPLES(NCITRIPLES) FIXED INITIAL ( 197373, 197386, 197389, 197401,
   197422, 197423, 197427, 209657, 590595, 590602, 590605, 590617, 590633,
   590639, 590643, 602383, 656131, 656138, 656141, 656153, 656174, 656175,
   656179, 660419, 721667, 721674, 721677, 721689, 721710, 721711, 721715,
   733955, 787203, 787210, 787213, 787225, 787246, 787247, 787[illegible], 7[illegible],
   865027, 918275, 918282, 918285, 918297, 918318, 918319, 918[illegible]3, 920563,
   996099, 994430, 1180419, 1180426, 1180429, 1180441, 1180462, 1180463,
   1180467, 1192707, 1245955, 1245962, 1245965, 1245977, 12459[illegible], 1245993,
   1246003, 1258243, 1389315, 1392644, 1392646, 1394945, 15083[illegible], 1508106,
   1508109, 1508121, 1508142, 1508143, 1508147, 1520387, 15736[illegible], 1573642,
   1573645, 1573657, 1573678, 1573679, 1573685, 1585023, 1639171, 1639178,
   1639181, 1639193, 1639214, 1639215, 1639219, 1651459, 1704717, 1704714,
   1704717, 1704729, 1704750, 1704751, 1704755, 1716985, 1966851, 1966853,
   1966861, 1966373, 1966894, 1966895, 1966899, 1979139, 2044675, 2294531,
   2294533, 2294541, 2294553, 2294574, 2294575, 2294579, 230681[illegible], 2491139,
   2491146, 2491149, 2491151, 2491182, 2491183, 2491187, 2503427, 2687747,
   2687754, 2687757, 2687769, 2687790, 2687791, 2687795, 2700035, 3015202,
   3015204, 3015212, 3157505, 3408643, 3408650, 3408653, 34086[illegible], 3408686,
   3408687, 3408691, 3420931, 3474179, 3474186, 3474189, 34742[illegible], 3474222,
   3474223, 3474227, 3486467, 3801859, 3801866, 3801869, 38018[illegible], 3801902,
   3801903, 3801907, 3814147, 4064003, 4064010, 4064013, 4064025, 4064046,
   4064047, 4064051, 4076291, 4326147, 4326154, 4326157, 43261[illegible], 4326190,
   4326191, 4326195, 4338435, 4469507, 4472836, 4472839, 4475137, 4480285,
   4535043, 4536372, 4533374, 4540673, 4669698, 4669700, 4731651, 4734980,
   4734982, 4928259, 4931568, 4931590, 4933889, 4992514, 5259522, 5259524,
   5456130, 5456132, 5518383, 5521412, 5521414, 5523713, 5649155, 5652484,
   5652486, 5654735, 5716457, 5730993, 5845762, 5845764, 6030083, 6030090,
   6030093, 6030105, 6030126, 6030127, 6030131, 6042371, 6504124, 6504452,
   6504454, 6906753, 6947843, 6947891);
DECLARE PRTS(129) FIXED INITIAL (0, 5724214, 5713965, 22326, 22360, 3935,
   3918, 3921, 87, 15, 73, 57, 105, 98, 85, 94, 105, 27, 40, 42, 0, 18241,
   20545, 22835, 24658, 21313, 842, 23626, 33, 109, 45, 13, 51, 0, 0, 22835,
   18241, 24658, 20545, 21313, 23626, 64, 51, 46, 7, 8, 0, 0, 0, 7, 0, 16, 0,
   0, 0, 3458, 92, 0, 0, 0, 51, 0, 0, 0, 59, 0, 13106, 0, 99, 22, 59, 90, 0,
   0, 4992514, 103, 0, 27401, 27402, 27427, 27430, 10, 0, 22100, 99, 75,
   14347, 14348, 14362, 0, 31, 13, 0, 13, 0, 17477, 64, 75, 68, 0, 51, 72,
   3426969, 16446, 52, 58, 30, 41, 99, 0, 109, 0, 0, 26367, 26391, 0, 99, 0,
   25, 0, 0, 4210250, 23320, 0, 16402, 0, 27140, 43, 27458, 0);
DECLARE PROTR(129) BIT(8) INITIAL (0, 38, 39, 36, 37, 25, 26, 27, 35, 24, 6,
   7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 61, 78. 41, 72, 115, 118, 122, 62,
   70, 79, 119, 123, 42, 73, 43, 63, 74, 80. 120, 124, 84, 47, 48, 101, 102,
   96, 83, 97, 100, 98, 54, 127, 128, 44, 21, 22, 55, 67, 69, 77, 129, 49,
   68, 53, 126, 50, 125, 40, 45, 52, 76, 75, 121, 65, 64, 99, 104, 105, 106,
   107, 109, 103, 34, 46, 23, 110, 111, 112, 108, 51, 117, 116, 114, 113, 19,
   3, 28, 18, 2, 60, 82, 31, 81, 30, 32, 33, 50, 20, 5, 66, 71, 1, 88, 89,
   87, 17, 4, 93, 92, 58, 29, 91, 90, 86, 85, 57, 56, 95, 94);
DECLARE HOTS(129) BIT(8) INITIAL (0, 86, 86, 86, 86, 75, 75, 75, 86, 75, 93,
   93, 93, 93, 93, 93, 93, 93, 93, 93, 93, 70, 79, 88, 108, 63, 63, 64, 71,
   76, 90, 83, 92, 69, 46, 80, 71, 96, 80, 83, 92, 72, 99, 99, 55, 55, 66,
   62, 66, 55, 66, 59, 57, 53, 60, 63, 69, 59, 54, 54, 90, 58, 58, 54, 94,
   65, 104, 65, 97, 60, 94, 82, 82, 64, 100, 100, 66, 107, 107, 107, 107,
   107, 107, 105, 60, 57, 56, 56, 55, 56, 85, 63, 63, 63, 63, 77, 84, 75, 77,
   84, 104, 73, 101, 73, 101, 78, 81, 98, 77, 67, 100, 108, 61, 103, 103,
   103, 93, 67, 102, 102, 104, 95, 91, 91, 74, 74, 106, 106, 97, 97);
DECLARE PRLENGTH(129) BIT(3) INITIAL (0, 4, 4, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2,
   2, 2, 2, 2, 2, 2, 2, 1, 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 1, 1, 3, 3, 3,
   3, 3, 3, 2, 2, 2, 2, 2, 1, 1, 1, 2, 1, 2, 1, 1, 1, 3, 2, 1, 1, 1, 2, 1, 1,
   1, 2, 1, 3, 1, 2, 2, 2, 2, 1, 1, 4, 2, 1, 3, 3, 3, 3, 2, 1, 3, 2, 2, 3, 3,
   3, 1, 2, 2, 1, 2, 1, 3, 2, 2, 2, 1, 2, 2, 4, 3, 2, 2, 2, 2, 2, 1, 2, 1, 1,
   3, 3, 1, 2, 1, 2, 1, 1, 4, 3, 1, 3, 1, 3, 2, 3, 1);
DECLARE CONTEXT_CASE(129) BIT(8) INITIAL (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      [illegible]
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0):
   DECLARE LEFT_CONTEXT(4) BIT(8) INITIAL ( 107, 4, 43, 96, 87);
   DECLARE LEFT_INDEX(57) BIT(8) INITIAL ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4, 4, 4, 4, 4, 4,
      4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5);
   DECLARE CONTEXT_TRIPLE(0) FIXED INITIAL ( 0);
   DECLARE TRIPLE_INDEX(57) BIT(8) INITIAL ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1);
   DECLARE PR_INDEX(108) BIT(8) INITIAL ( 1, 21, 29, 35, 42, 44, 45, 49, 51, 51,
      51, 51, 51, 51, 51, 51, 51, 53, 53, 53, 54, 54, 55, 55, 55, 55, 55, 55,
      56, 57, 57, 57, 58, 58, 59, 59, 60, 61, 61, 62, 62, 63, 63, 63, 64, 64,
      66, 69, 69, 69, 69, 74, 74, 74, 76, 77, 83, 83, 83, 83, 86, 86, 86, 90,
      93, 95, 95, 100, 100, 100, 101, 101, 101, 102, 108, 108, 108, 110, 110,
      111, 111, 111, 112, 112, 113, 113, 113, 113, 113, 113, 113, 116, 116, 118,
      118, 118, 118, 120, 120, 120, 121, 122, 124, 126, 128, 128, 128, 130,
      130);

   /*  END OF CARDS PUNCHED BY SYNTAX                                        */

   /*  DECLARATIONS FOR THE SCANNER                                          */

   /* TOKEN IS THE INDEX INTO THE VOCABULARY V() OF THE LAST SYMBOL SCANNED,
      CP IS THE POINTER TO THE LAST CHARACTER SCANNED IN THE CARDIMAGE,
      BCD IS THE LAST SYMBOL SCANNED (LITERAL CHARACTER STRING). */
   DECLARE (TOKEN, CP) FIXED, BCD CHARACTER;

   /* SET UP SOME CONVENIENT ABBREVIATIONS FOR PRINTER CONTROL */
   DECLARE EJECT_PAGE LITERALLY 'OUTPUT(1) = PAGE',
      PAGE CHARACTER INITIAL ('1'), DOUBLE CHARACTER INITIAL ('0'),
      DOUBLE_SPACE LITERALLY 'OUTPUT(1) = DOUBLE',
      X70 CHARACTER INITIAL ('
                        ');

   /* LENGTH OF LONGEST SYMBOL IN V */
   DECLARE (RESERVED_LIMIT, MARGIN_CHOP) FIXED;

   /* CHARTYPE() IS USED TO DISTINGUISH CLASSES OF SYMBOLS IN THE SCANNER.
      TX() IS A TABLE USED FOR TRANSLATING FROM ONE CHARACTER SET TO ANOTHER.
      CONTROL() HOLDS THE VALUE OF THE COMPILER CONTROL TOGGLES SET IN $ CARDS.
      NOT_LETTER_OR_DIGIT() IS SIMILIAR TO CHARTYPE() BUT USED IN SCANNING
      IDENTIFIERS ONLY.

      ALL ARE USED BY THE SCANNER AND CONTROL() IS SET THERE.
   */
   DECLARE (CHARTYPE, TX) (255) BIT(8),
           (CONTROL, NOT_LETTER_OR_DIGIT)(255) BIT(1);

   /* ALPHABET CONSISTS OF THE SYMBOLS CONSIDERED ALPHABETIC IN BUILDING
      IDENTIFIERS     */
   DECLARE ALPHABET CHARACTER INITIAL ('ABCDEFGHIJKLMNOPQRSTUVWXYZ_$@#');

   /* BUFFER HOLDS THE LATEST CARDIMAGE,
      TEXT HOLDS THE PRESENT STATE OF THE INPUT TEXT
      (NOT INCLUDING THE PORTIONS DELETED BY THE SCANNER),
      TEXT_LIMIT IS A CONVENIENT PLACE TO STORE THE POINTER TO THE END OF TEXT,
      CARD_COUNT IS INCREMENTED BY ONE FOR EVERY SOURCE CARD READ,
      ERROR_COUNT TABULATES THE ERRORS AS THEY ARE DETECTED,
      SEVERE_ERRORS TABULATES THOSE ERRORS OF FATAL SIGNIFICANCE.
   */
   DECLARE (BUFFER, TEXT) CHARACTER,
      (TEXT_LIMIT, CARD_COUNT, ERROR_COUNT, SEVERE_ERRORS, PREVIOUS_ERROR) FIXED
      ;
```

```

/* NUMBER_VALUE CONTAINS THE NUMERIC VALUE OF THE LAST CONSTANT SCANNED,
*/
DECLARE NUMBER_VALUE FIXED;

/* EACH OF THE FOLLOWING CONTAINS THE INDEX INTO V() OF THE CORRESPONDING
   SYMBOL.   WE ASK:     IF TOKEN = IDENT     ETC.     */
DECLARE (IDENT, NUMBER, DIVIDE, EOFILE) FIXED;

/* STOPIT() IS A TABLE OF SYMBOLS WHICH ARE ALLOWED TO TERMINATE THE ERROR
   FLUSH PROCESS.  IN GENERAL THEY ARE SYMBOLS OF SUFFICIENT SYNTACTIC
   HIERARCHY THAT WE EXPECT TO AVOID ATTEMPTING TO START CHECKING AGAIN
   RIGHT INTO ANOTHER ERROR PRODUCING SITUATION.  THE TOKEN STACK IS ALSO
   FLUSHED DOWN TO SOMETHING ACCEPTABLE TO A STOPIT() SYMBOL.
   FAILSOFT IS A BIT WHICH ALLOWS THE COMPILER ONE ATTEMPT AT A GENTLE
   RECOVERY.   THEN IT TAKES A STRONG HAND.   WHEN THERE IS REAL TROUBLE
   COMPILING IS SET TO FALSE, THEREBY TERMINATING THE COMPILATION.
*/
DECLARE STOPIT(100) BIT(1), (FAILSOFT, COMPILING) BIT(1);

DECLARE S CHARACTER;  /* A TEMPORARY USED VARIOUS PLACES */

/* THE ENTRIES IN PRMASK() ARE USED TO SELECT OUT PORTIONS OF CODED
   PRODUCTIONS AND THE STACK TOP FOR COMPARISON IN THE ANALYSIS ALGORITHM */
DECLARE PRMASK(5) FIXED INITIAL (0, 0, "FF", "FFFF", "FFFFFF", "FFFFFFFF");


/*THE PROPER SUBSTRING OF POINTER IS USED TO PLACE AN      UNDER THE POINT
   OF DETECTION OF AN ERROR DURING CHECKING.  IT MARKS THE LAST CHARACTER
   SCANNED.  */
DECLARE POINTER CHARACTER INITIAL ('
                                                   ');
DECLARE CALLCOUNT(20) FIXED   /* COUNT THE CALLS OF IMPORTANT PROCEDURES */
   INITIAL(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);

/* RECORD THE TIMES OF IMPORTANT POINTS DURING CHECKING */
DECLARE CLOCK(5) FIXED;


/* COMMONLY USED STRINGS */
DECLARE X1 CHARACTER INITIAL(' '), X4 CHARACTER INITIAL('    ');
DECLARE PERIOD CHARACTER INITIAL ('.');

/* TEMPORARIES USED THROUGHOUT THE COMPILER */
DECLARE (I, J, K, L) FIXED;

DECLARE TRUE LITERALLY '1', FALSE LITERALLY '0', FOREVER LITERALLY 'WHILE 1';

/*  THE STACKS DECLARED BELOW ARE USED TO DRIVE THE SYNTACTIC
   ANALYSIS ALGORITHM AND STORE INFORMATION RELEVANT TO THE INTERPRETATION
   OF THE TEXT.  THE STACKS ARE ALL POINTED TO BY THE STACK POINTER SP.  */

DECLARE STACKSIZE LITERALLY '75';  /* SIZE OF STACK  */
DECLARE PARSE_STACK (STACKSIZE) BIT(3); /* TOKENS OF THE PARTIALLY PARSED
                                           TEXT */
DECLARE VAR (STACKSIZE) CHARACTER;/* EBCDIC NAME OF ITEM */
DECLARE FIXV (STACKSIZE) FIXED;   /* FIXED (NUMERIC) VALUE */

/* SP POINTS TO THE RIGHT END OF THE REDUCIBLE STRING IN THE PARSE STACK,
   MP POINTS TO THE LEFT END, AND
   MPP1 = MP+1.
*/
DECLARE (SP, MP, MPP1) FIXED;

  DECLARE SAVEREF FIXED; /*  REFERS BACK TO A PREVIOUSLY
                             REFERENCED LABEL  */
```

```
      DECLARE NLABEL FIXED INITIAL(0);
      DECLARE LABELS LITERALLY '25';  /*  SIZE OF LABEL TABLE  */
      DECLARE LABID(LABELS) CHARACTER;  /*  LABEL IDENTIFIER  */
      DECLARE (LASTREF,LABDEF) (LABELS) FIXED;  /* REST OF LABEL TABLE */
      DECLARE TBASIC FIXED;
      DECLARE SAVEQUAD FIXED,
              SAVEQUAD2 FIXED;
      DECLARE NEXTQUAD FIXED INITIAL (1);
      DECLARE TABLE_LOC (STACKSIZE) FIXED;    /* LOCATION OF SYMBOLS IN TABLES */
      DECLARE SYMBOLS LITERALLY '100'; /* SIZE OF SYMBOL TABLE */
      DECLARE SYMB(SYMBOLS) CHARACTER ; /* IDENTIFIER */
      DECLARE (LOCAT,DEF,SIZE,INIT) (SYMBOLS) FIXED; /* REST OF SYMBOL TABLE */
      DECLARE SAVQUD(10) FIXED;
      DECLARE SAVQUDNO FIXED INITIAL(0);
      DECLARE SAVLAB(10) CHARACTER;
      DECLARE SAVLABNO FIXED INITIAL(0);
 /*    QUAD TYPES */
      DECLARE NVARDEF FIXED,
              NSYMBOL FIXED INITIAL (0),
              NCONSTANT FIXED INITIAL (0),
      ADD   FIXED  INITIAL  (1),
      MUL   FIXED  INITIAL  (2),
      SUB   FIXED  INITIAL  (3),
      DIV   FIXED  INITIAL  (4),
      MMOD  FIXED  INITIAL  (5),
      HALT  FIXED  INITIAL  (6),
      BR    FIXED  INITIAL  (7),
      BT    FIXED  INITIAL  (8),
      BF    FIXED  INITIAL  (9),
      REL   FIXED  INITIAL  (10),
      EQ    FIXED  INITIAL  (11),
      LT    FIXED  INITIAL  (12),
      GT    FIXED  INITIAL  (13),
      NE    FIXED  INITIAL  (14),
      LE    FIXED  INITIAL  (15),
      GE    FIXED  INITIAL  (16),
      ASGN  FIXED  INITIAL  (17),
      SUBS  FIXED  INITIAL  (18),
      BZ    FIXED  INITIAL  (19),
      AND   FIXED  INITIAL  (20),
      OR    FIXED  INITIAL  (21),
      UMIN  FIXED  INITIAL  (22),
      ZQ    FIXED  INITIAL  (23),
      LAB   FIXED  INITIAL  (24),
      SUBL  FIXED  INITIAL  (30),                                  /*NEWQUAD*/
      TGOTO  FIXED  INITIAL (0);
      DECLARE CORE FIXED INITIAL (0);
      DECLARE SAVEINDEX FIXED;
      DECLARE LOOPLIM FIXED INITIAL (0),
              LOOPINC FIXED INITIAL (1),
              LOOP_INDEX FIXED INITIAL (0);
      DECLARE SAVELOC FIXED;
      DECLARE SAVELAB CHARACTER;
      DECLARE SAVEVAR CHARACTER;
      DECLARE DO_SWITCH FIXED INITIAL (0);
      DECLARE (MULLOC,DIVLOC) FIXED;
      DECLARE CONSTANTS LITERALLY '50'; /* SIZE OF CONSTANTS TABLE */
      DECLARE (CONLOC,CONVAL) (CONSTANTS) FIXED; /* CONSTANTS TABLE */
  /*   DISK BUFFER FOR QUAD TABLE          */
      DECLARE DISKWORDS LITERALLY '900',
              MAXQUADS LITERALLY '220',
              QUADS (DISKWORDS) FIXED;
      DECLARE RCD_BUFF FIXED INITIAL (0);
      DECLARE RCD_NR FIXED;
      DECLARE LOC_QUAD FIXED;
          DECLARE BR_FLAG BIT(8);                                  /*NEWQUAD*/
```

```
|        DECLARE CURR_SIN FIXED INITIAL(0);                           /*NEWQUAD*/ | 12
|        DECLARE MIC_LOC(25) FIXED;                                   /*NEWQUAD*/ | 12
|        DECLARE BUF_MIC CHARACTER INITIAL(                           /*NEWQUAD*/ | 12
|'000 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 00000000');    | 12
|        DECLARE(OPERATION,OPERAND1,OPERAND2,RESULT)CHARACTER;        /*NEWQUAD*/ | 12
|        DECLARE (LAVS) FIXED INITIAL(0);                             /*NEWQUAD*/ | 12
|        DECLARE VARIBLS(40) CHARACTER;                               /*NEWQUAD*/ | 12
|        DECLARE CHANGE(40) BIT(8);                                   /*NEWQUAD*/ | 12
|        DECLARE NEXT_VAR(40) FIXED INITIAL                           /*NEWQUAD*/ | 128
|        (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,   /*NEWQUAD*/ | 128
|        23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40);      /*NEWQUAD*/ | 128
|        DECLARE (RD) BIT(8);                                         /*NEWQUAD*/ | 128
|        DECLARE STATUS(7) BIT(8);                                    /*NEWQUAD*/ | 128
|    DECLARE ADDRSS(7) BIT(8);                                        /*NEWQUAD*/ | 128
|    DECLARE SUBSFLAG FIXED INITIAL(0);                               /*NEWQUAD*/ | 128
|        DECLARE VARNUM(7) FIXED;                                     /*NEWQUAD*/ | 128
|        DECLARE DEALLOCABL(7) BIT(8);                                /*NEWQUAD*/ | 128
|        DECLARE REFRENCE(7) FIXED;                                   /*NEWQUAD*/ | 128
|        DECLARE TEMP(7) FIXED;                                       /*NEWQUAD*/ | 1286
|        DECLARE POINT(7) FIXED;                                      /*NEWQUAD*/ | 1286
|        DECLARE MAXREG FIXED INITIAL(6);                             /*NEWQUAD*/ | 1286
|        DECLARE(II,JJ,KK,LL,NN)FIXED;                                /*NEWQUAD*/ | 1286
|        DECLARE (REFNO) FIXED INITIAL (0);                           /*NEWQUAD*/ | 1286
|        DECLARE SORTREF(6) FIXED;                                    /*NEWQUAD*/ | 1286
|        DECLARE SORTNUM(6) FIXED;                                    /*NEWQUAD*/ | 1286
|        DECLARE REGNO(6) FIXED;                                      /*NEWQUAD*/ | 1286
|        DECLARE NEWQUAD(900) FIXED;                                  /*NEWQUAD*/ | 1286
|        DECLARE NEWQUADNO FIXED INITIAL(0);                          /*NEWQUAD*/ | 1286
|        DECLARE TEMP_CHAR CHARACTER;                                 /*NEWQUAD*/ | 1286
|        DECLARE(OPRTOR,OPRND1,OPRND2,RSLT)FIXED;                     /*NEWQUAD*/ | 1286
|        DECLARE OPER FIXED;                                          /*NEWQUAD*/ | 1286
|    DECLARE ALPHA(30) CHARACTER INITIAL ('WTAD',                     /*NEWQUAD*/ | 1286
|                                          'ADD ',                    /*NEWQUAD*/ | 1286
|                                          'MUL ',                    /*NEWQUAD*/ | 1286
|                                          'SUB ',                    /*NEWQUAD*/ | 1286
|                                          'DIV ',                    /*NEWQUAD*/ | 1286
|                                          'MOD ',                    /*NEWQUAD*/ | 1286
|                                          'HALT',                    /*NEWQUAD*/ | 1286
|                                          'BR  ',                    /*NEWQUAD*/ | 1286
|                                          'BT  ',                    /*NEWQUAD*/ | 1286
|                                          'BF  ',                    /*NEWQUAD*/ | 1286
|                                          'REL ',                    /*NEWQUAD*/ | 1286
|                                          'EQ  ',                    /*NEWQUAD*/ | 1286
|                                          'LT  ',                    /*NEWQUAD*/ | 1286
|                                          'GT  ',                    /*NEWQUAD*/ | 1286
|                                          'NE  ',                    /*NEWQUAD*/ | 1286
|                                          'LE  ',                    /*NEWQUAD*/ | 1286
|                                          'GE  ',                    /*NEWQUAD*/ | 1286
|                                          'ASGN',                    /*NEWQUAD*/ | 1286
|                                          'SUBS',                    /*NEWQUAD*/ | 1286
|                                          'BZ  ',                    /*NEWQUAD*/ | 1286
|                                          'AND ',                    /*NEWQUAD*/ | 1286
|                                          'OR  ',                    /*NEWQUAD*/ | 1286
|                                          'UMIN',                    /*NEWQUAD*/ | 1286
|                                          'ZQ  ',                    /*NEWQUAD*/ | 1286
|                                          'LAB ',                    /*NEWQUAD*/ | 1286
|                                          'RD  ',                    /*NEWQUAD*/ | 1286
|                                          'WT  ',                    /*NEWQUAD*/ | 1286
|                                          'RDAD',                    /*NEWQUAD*/ | 1286
|                                          'RDVR',                    /*NEWQUAD*/ | 1286
|                                          'ASL ',                    /*NEWQUAD*/ | 1286
|                                      'SUBL');                       /*NEWQUAD*/ | 1286
|                                                                                 | 1286
|                                                                                 | 1286
|                                                                                 | 1286
|                                                                                 | 1286
|                                                                      |  1286
|  /*              P R O C E D U R E S                            */   |  1286
|                                                                      |  1286
|                                                                      |  1286
|                                                                      |  1286
|PAD:                                                                  |  1286
|   PROCEDURE (STRING, WIDTH) CHARACTER;                               |  1286
|      DECLARE STRING CHARACTER, (WIDTH, L) FIXED;                     |  1286 PAD
|                                                                      |  1294 PAD
|      L = LENGTH(STRING);                                             |  1294 PAD
|      IF L >= WIDTH THEN RETURN STRING;                               |  1316 PAD
|      ELSE RETURN STRING || SUBSTR(X70, 0, WIDTH-L);                  |  1338 PAD
|   END PAD;                                                           |  1390 PAD
|                                                                      |  1396
|I_FORMAT:                                                             |  1396
|   PROCEDURE (NUMBER, WIDTH) CHARACTER;                               |  1396
|      DECLARE (NUMBER, WIDTH, L) FIXED, STRING CHARACTER;             |  1396 I_FORMA
|                                                                      |  1404 I_FORMA
|      STRING = NUMBER;                                                |  1404 I_FORMA
|      L = LENGTH(STRING);                                             |  1420 I_FORMA
|      IF L >= WIDTH THEN RETURN STRING;                               |  1442 I_FORMA
|      ELSE RETURN SUBSTR(X70, 0, WIDTH-L) || STRING;                  |  1464 I_FORMA
|   END I_FORMAT;                                                      |  1516 I_FORMA
|                                                                      |  1522
|ERROR:                                                                |  1522
|   PROCEDURE(MSG, SEVERITY);                                          |  1522
|      /* PRINTS AND ACCOUNTS FOR ALL ERROR MESSAGES */                |  1522 ERROR
|      /* IF SEVERITY IS NOT SUPPLIED, 0 IS ASSUMED */                 |  1522 ERROR
|      DECLARE MSG CHARACTER, SEVERITY FIXED;                          |  1522 ERROR
|      ERROR_COUNT = ERROR_COUNT + 1;                                  |  1530 ERROR
|      /* IF LISTING IS SUPPRESSED, FORCE PRINTING OF THIS LINE */     |  1542 ERROR
|      IF ¬ CONTROL(BYTE('L')) THEN                                    |  1542 ERROR
|         OUTPUT = I_FORMAT (CARD_COUNT, 4) || ' |' || BUFFER || '|';  |  1556 ERROR
|      OUTPUT = SUBSTR(POINTER, TEXT_LIMIT-CP+MARGIN_CHOP);            |  1650 ERROR
|      OUTPUT = '*** ERROR, ' || MSG ||                                |  1698 ERROR
|            '.  LAST PREVIOUS ERROR WAS DETECTED ON LINE ' ||         |  1718 ERROR
|            PREVIOUS_ERROR || '.  ***';                               |  1734 ERROR
|      PREVIOUS_ERROR = CARD_COUNT;                                    |  1792 ERROR
|      IF SEVERITY > 0 THEN                                            |  1800 ERROR
|         IF SEVERE_ERRORS > 25 THEN                                   |  1820 ERROR
|            DO;                                                       |  1832 ERROR
|               OUTPUT = '*** TOO MANY SEVERE ERRORS, CHECKING ABORTED ***';  |  1824 ERROR
|               COMPILING = FALSE;                                     |  1844 ERROR
|            END;                                                      |  1850 ERROR
|         ELSE SEVERE_ERRORS = SEVERE_ERRORS + 1;                      |  1850 ERROR
|   END ERROR;                                                         |  1866 ERROR
|                                                                      |  1872
|                                                                      |  1872
|                                                                      |  1872
|                                                                      |  1872
|                                                                      |  1872
|                                                                      |  1872
| /*                     CARD IMAGE HANDLING PROCEDURE                */  |  1872
|                                                                      |  1872
|                                                                      |  1872
|GET_CARD:                                                             |  1872
|   PROCEDURE;                                                         |  1872
|      /* DOES ALL CARD READING AND LISTING                          */ |  1872 GET_CAR
|      DECLARE I FIXED, (TEMP, TEMPO, REST) CHARACTER, READING BIT(1); |  1872 GET_CAR
|            BUFFER = INPUT;                                           |  1880 GET_CAR
|            IF LENGTH(BUFFER) = 0 THEN                                |  1942 GET_CAR
|               DO; /* SIGNAL FOR EOF */                               |  1976 GET_CAR
|                  CALL ERROR ('EOF MISSING OR COMMENT STARTING IN COLUMN 1.',1);|  1968 GET_CAR
|                  BUFFER = PAD (' /*''/* */ EOF;END;EOF', 80);        |  1988 GET_CAR
|               END;                                                   |  2012 GET_CAR
|            ELSE CARD_COUNT = CARD_COUNT + 1;  /* USED TO PRINT ON LISTING */  |  2012 GET_CAR
```

```
      IF MARGIN_CHOP > 0 THEN                                                    |  2028 GET_CARD
         DO; /* THE MARGIN CONTROL FROM DOLLAR    */                             |  2048 GET_CARD
            I = LENGTH(BUFFER) - MARGIN_CHOP;                                    |  2060 GET_CARD
            REST = SUBSTR(BUFFER, I);                                            |  2066 GET_CARD
            BUFFER = SUBSTR(BUFFER, 0, I);                                       |  2092 GET_CARD
         END;                                                                    |  2118 GET_CARD
      ELSE REST = '';                                                            |  2118 GET_CARD
      TEXT = BUFFER;                                                             |  2128 GET_CARD
      TEXT_LIMIT = LENGTH(TEXT) - 1;                                             |  2136 GET_CARD
      IF CONTROL(BYTE('M')) THEN OUTPUT = BUFFER;                                |  2162 GET_CARD
      ELSE IF CONTROL(BYTE('L')) THEN                                            |  2196 GET_CARD
         OUTPUT = I_FORMAT (CARD_COUNT, 4)  ||' |' || BUFFER || '|' || REST;     |  2204 GET_CARD
      CP = 0;                                                                    |  2316 GET_CARD
   END GET_CARD;                                                                 |  2322 GET_CARD
                                                                                 |  2328
                                                                                 |  2328
   /*                   THE SCANNER PROCEDURES                    */             |  2328
                                                                                 |  2328
                                                                                 |  2328
CHAR:                                                                            |  2328
   PROCEDURE;                                                                    |  2328
      /* USED FOR STRINGS TO AVOID CARD BOUNDARY PROBLEMS */                     |  2328 CHAR
      CP = CP + 1;                                                               |  2328 CHAR
      IF CP <= TEXT_LIMIT THEN RETURN;                                           |  2348 CHAR
      CALL GET_CARD;                                                             |  2366 CHAR
   END CHAR;                                                                     |  2370 CHAR
                                                                                 |  2376
                                                                                 |  2376
SCAN:                                                                            |  2376
   PROCEDURE;                                                                    |  2376
      DECLARE (S1, S2) FIXED;                                                    |  2376 SCAN
      CALLCOUNT(3) = CALLCOUNT(3) + 1;                                           |  2384 SCAN
      FAILSOFT = TRUE;                                                           |  2412 SCAN
      BCD = '';  NUMBER_VALUE = 0;                                               |  2420 SCAN.
   SCAN1:                                                                        |  2432 SCAN
      DO FOREVER;                                                                |  2432 SCAN
         IF CP > TEXT_LIMIT THEN CALL GET_CARD;                                  |  2432 SCAN
         ELSE                                                                    |  2448 SCAN
            DO; /* DISCARD LAST SCANNED VALUE */                                 |  2448 SCAN
               TEXT_LIMIT = TEXT_LIMIT - CP;                                     |  2452 SCAN
               TEXT = SUBSTR(TEXT, CP);                                          |  2464 SCAN
               CP = 0;                                                           |  2490 SCAN
            END;                                                                 |  2496 SCAN
         /*  BRANCH ON NEXT CHARACTER IN TEXT                     */             |  2496 SCAN
         DO CASE CHARTYPE(BYTE(TEXT));                                           |  2496 SCAN
                                                                                 |  2528 SCAN
            /*  CASE 0  */                                                       |  2528 SCAN
                                                                                 |  2528 SCAN
            /* ILLEGAL CHARACTERS FALL HERE  */                                  |  2528 SCAN
            CALL ERROR ('ILLEGAL CHARACTER: ' || SUBSTR(TEXT, 0, 1));            |  2528 SCAN
                                                                                 |  2564 SCAN CASE 0.
            /*  CASE 1  */                                                       |  2564 SCAN
                                                                                 |  2564 SCAN
            /*  BLANK  */                                                        |  2564 SCAN
            DO;                                                                  |  2564 SCAN
               CP = 1;                                                           |  2568 SCAN CASE 1.
               DO WHILE BYTE(TEXT, CP) = BYTE(' ') & CP <= TEXT_LIMIT;           |  2576 SCAN
                  CP = CP + 1;                                                   |  2634 SCAN C8 = 64
               END;                                                              |  2646 SCAN
               CP = CP - 1;                                                      |  2650 SCAN
            END;                                                                 |  2662 SCAN
                                                                                 |  2662 SCAN
            /*  CASE 2  */                                                       |  2662 SCAN
                                                                                 |  2662 SCAN
          ; /*  NOT USED IN SKELETON (BUT USED IN XCOM)  */                      |  2662 SCAN
                                                                                 |  2666 SCAN CASE 2.
```

```
    /*  CASE 3  */                                                             |  2666 SCAN
                                                                               |  2666 SCAN
;   /*  NOT USED IN SKELETON (BUT USED IN XCOM)  */                            |  2666 SCAN
                                                                               |  2670 SCAN CASE 3
    /*  CASE 4  */                                                             |  2670 SCAN
                                                                               |  2670 SCAN
    DO FOREVER;  /* A LETTER:  IDENTIFIERS AND RESERVED WORDS */               |  2670 SCAN
        DO CP = CP + 1 TO TEXT_LIMIT;                                          |  2674 SCAN CASE 4
            IF NOT_LETTER_OR_DIGIT(BYTE(TEXT, CP)) THEN                        |  2714 SCAN
                DO;  /* END OF IDENTIFIER  */                                  |  2726 SCAN
                    IF CP > 0 THEN BCD = BCD || SUBSTR(TEXT, 0, CP);           |  2736 SCAN
                    S1 = LENGTH(BCD);                                          |  2790 SCAN
                    IF S1 > 1 THEN IF S1 <= RESERVED_LIMIT THEN                |  2812 SCAN
                        /* CHECK FOR RESERVED WORDS */                         |  2844 SCAN
                        DO I = V_INDEX(S1-1) TO V_INDEX(S1) - 1;               |  2844 SCAN
                            IF BCD = V(I) THEN                                 |  2892 SCAN
                                DO;                                            |  2942 SCAN
                                    TOKEN = I;                                 |  2934 SCAN
                                    RETURN;                                    |  2942 SCAN
                                END;                                           |  2948 SCAN
                        END;                                                   |  2948 SCAN
                    /*  RESERVED WORDS EXIT HIGHER: THEREFORE <IDENTIFIER>*/   |  2952 SCAN
                    TOKEN = IDENT;                                             |  2952 SCAN
                    RETURN;                                                    |  2960 SCAN
                END;                                                           |  2966 SCAN
        END;                                                                   |  2966 SCAN
        /*  END OF CARD  */                                                    |  2970 SCAN
        BCD = BCD || TEXT;                                                     |  2970 SCAN
        CALL GET_CARD;                                                         |  2994 SCAN
        CP = -1;                                                               |  2998 SCAN
    END;                                                                       |  3006 SCAN
                                                                               |  3010 SCAN
                                                                               |  3010 SCAN
    /*  CASE 5  */                                                             |  3010 SCAN
                                                                               |  3010 SCAN
    DO;        /*  DIGIT:  A NUMBER  */                                        |  3010 SCAN
        TOKEN = NUMBER;                                                        |  3014 SCAN CASE 5
        DO FOREVER;                                                            |  3022 SCAN
            DO CP = CP TO TEXT_LIMIT;                                          |  3022 SCAN
                S1 = BYTE(TEXT, CP);                                           |  3058 SCAN
                IF S1 < "F0" THEN RETURN;                                      |  3074 SCAN
                NUMBER_VALUE = 10*NUMBER_VALUE + S1 - "F0";                    |  3092 SCAN
            END;                                                               |  3112 SCAN
            CALL GET_CARD;                                                     |  3116 SCAN
        END;                                                                   |  3120 SCAN
    END;                                                                       |  3124 SCAN
                                                                               |  3124 SCAN
    /*  CASE 6  */                                                             |  3124 SCAN
                                                                               |  3124 SCAN
    DO;        /*  A /:  MAY BE DIVIDE OR START OF COMMENT  */                 |  3124 SCAN
        CALL CHAR;                                                             |  3128 SCAN CASE 6
        IF BYTE(TEXT, CP) ¬= BYTE('*') THEN                                    |  3132 SCAN
            DO;                                                                |  3160 SCAN
                TOKEN = DIVIDE;                                                |  3152 SCAN
                RETURN;                                                        |  3160 SCAN
            END;                                                               |  3166 SCAN
        /* WE HAVE A COMMENT  */                                               |  3166 SCAN
        S1, S2 = BYTE(' ');                                                    |  3166 SCAN
        DO WHILE S1 ¬= BYTE('*') | S2 ¬= BYTE('/');                            |  3178 SCAN
            IF S1 = BYTE('$') THEN                                             |  3228 SCAN
                DO;  /* A CONTROL CHARACTER  */                                |  3248 SCAN
                    CONTROL(S2) = ¬ CONTROL(S2);                               |  3240 SCAN
                    IF S2 = BYTE('T') THEN CALL TRACE;                         |  3262 SCAN
                    ELSE IF S2 = BYTE('U') THEN CALL UNTRACE;                  |  3268 SCAN
                    ELSE IF S2 = BYTE('|') THEN                                |  3318 SCAN
                        IF CONTROL(S2) THEN                                    |  3342 SCAN
```

```
|                     MARGIN_CHOP = TEXT_LIMIT - CP + 1;                      | 3356 SCAN
|                  ELSE                                                       | 3364 SCAN
|                     MARGIN_CHOP = 0;                                        | 3364 SCAN
|              END;                                                           | 3374 SCAN
|           S1 = S2;                                                          | 3374 SCAN
|           CALL CHAR;                                                        | 3382 SCAN
|           S2 = BYTE(TEXT, CP);                                              | 3386 SCAN
|         . END;                                                              | 3402 SCAN
|        END;                                                                 | 3406 SCAN
|                                                                             | 3406 SCAN
|        /*  CASE 7  */                                                       | 3406 SCAN
|        DO;        /*  SPECIAL CHARACTERS  */                                | 3406 SCAN
|           TOKEN = TX(BYTE(TEXT));                                           | 3410 SCAN CASE 7
|           CP = 1;                                                           | 3430 SCAN
|           RETURN;                                                           | 3438 SCAN
|      _. END;                                                                | 3444 SCAN
|                                                                             | 3444 SCAN
|        /*  CASE 8  */                                                       | 3444 SCAN
|     ;  /*  NOT USED IN SKELETON (BUT USED IN XCOM)  */                      | 3444 SCAN
|                                                                             | 3448 SCAN CASE 8
|      END;     /* OF CASE ON CHARTYPE  */                                    | 3448 SCAN
|      CP = CP + 1;  /* ADVANCE SCANNER AND RESUME SEARCH FOR TOKEN  */       | 3452 SCAN
|    END;                                                                     | 3464 SCAN
|  END SCAN;                                                                  | 3468 SCAN
|                                                                             | 3474
|                                                                             | 3474
|                                                                             | 3474
|                                                                             | 3474
|                                                                             | 3474
|  /*                         TIME AND DATE                               */  | 3474
|                                                                             | 3474
|                                                                             | 3474
|PRINT_TIME:                                                                  | 3474
|  PROCEDURE (MESSAGE, T);                                                    | 3474
|     DECLARE MESSAGE CHARACTER, T FIXED;                                     | 3474 PRINT_TIME
|   . MESSAGE = MESSAGE || T/360000 || ':' || T MOD 360000 / 6000 || ':'      | 3482 PRINT_TIME
|        || T MOD 6000 / 100 || '.';                                          | 3580 PRINT_TIME
|     T = T MOD 100;  /* DECIMAL FRACTION  */                                 | 3662 PRINT_TIME
|     IF T < 10 THEN MESSAGE = MESSAGE || '0';                                | 3680 PRINT_TIME
|     OUTPUT = MESSAGE || T || '.';                                           | 3716 PRINT_TIME
|  END PRINT_TIME;                                                            | 3773 PRINT_TIME
|                                                                             | 3784
|PRINT_DATE_AND_TIME:                                                         | 3784
|  PROCEDURE (MESSAGE, D, T);                                                 | 3784
|     DECLARE MESSAGE CHARACTER, (D, T, YEAR, DAY, M) FIXED;                  | 3784 PRINT_DATE_
|     DECLARE MONTH(11) CHARACTER INITIAL ('JANUARY', 'FEBRUARY', 'MARCH',    | 3792 PRINT_DATE_
|        'APRIL', 'MAY', 'JUNE', 'JULY', 'AUGUST', 'SEPTEMBER', 'OCTOBER',    | 3792 PRINT_DATE_
|        'NOVEMBER', 'DECEMBER'),                                             | 3792 PRINT_DATE_
|     DAYS(12) FIXED INITIAL (0, 31, 60, 91, 121, 152, 182, 213, 244, 274,    | 3792 PRINT_DATE_
|        305, 335, 366);                                                      | 3792 PRINT_DATE_
|     YEAR = D/1000 + 1900;                                                   | 3792 PRINT_DATE_
|     DAY = D MOD 1000;                                                       | 3812 PRINT_DATE_
|     IF (YEAR & "3") ¬= 0 THEN IF DAY > 59 THEN DAY = DAY + 1; /*   LEAP YEAR*/| 3830 PRINT_DATE_
|     M = 1;                                                                  | 3870 PRINT_DATE_
|     DO WHILE DAY > DAYS(M);  M = M + 1;  END;                               | 3878 PRINT_DATE_
|     CALL PRINT_TIME(MESSAGE || MONTH(M-1) || X1 || DAY-DAYS(M-1) ||  ', '   | 3914 PRINT_DATE_
|        || YEAR || '.  CLOCK TIME = ', T);                                   | 4002 PRINT_DATE_
|  END PRINT_DATE_AND_TIME;                                                   | 4074 PRINT_DATE_
|                                                                             | 4080
|  /*                         INITIALIZATION                              */| | 4080
|                                                                             | 4080
|                                                                             | 4080
|                                                                             | 4080
|INITIALIZATION:                                                              | 4080
|  PROCEDURE;                                                                 | 4080
|     EJECT PAGE;                                                             | 4080 INITIALIZA
```

```
   CALL PRINT_DATE_AND_TIME ('   PL/M COMPILER FOR THE PDP11/20 - C05012 - BY RI   |   4112 INITIALIZA
WEST AND OF WILLIS ON ',DATE_OF_GENERATION,TIME_OF_GENERATION);                    |   4112 INITIALIZA
      DOUBLE_SPACE;                                                                |   4140 INITIALIZA
      CALL PRINT_DATE_AND_TIME ('TODAY IS ', DATE, TIME);                          |   4164 INITIALIZA
      DOUBLE_SPACE;                                                                |   4214 INITIALIZA
     DO II=1 TO MAXREG;                                              /*NEWQUAD*/   |   4238 INITIALIZA
        STATUS(II) = FALSE;                                          /*NEWQUAD*/   |   4282 INITIALIZA
        ADDRSS(II)=FALSE;                                            /*NEWQUAD*/   |   4292 INITIALIZA
        END;                                                         /*NEWQUAD*/   |   4302 INITIALIZA
      DO I = 1 TO NT;                                                              |   4310 INITIALIZA
        S = V(I);                                                                  |   4346 INITIALIZA
        IF S = '<NUMBER>' THEN NUMBER = I;  ELSE                                   |   4362 INITIALIZA
        IF S = '<IDENTIFIER>' THEN IDENT = I;  ELSE                                |   4412 INITIALIZA
        IF S = '/' THEN DIVIDE = I;  ELSE                                          |   4470 INITIALIZA
        IF S = '_|_' THEN EOFILE = I;  ELSE                                        |   4528 INITIALIZA
        IF S = ';' THEN STOPIT(I) = TRUE;  ELSE                                    |   4586 INITIALIZA
        ;                                                                          |   4648 INITIALIZA
      END;                                                                         |   4656 INITIALIZA
      IF IDENT = NT THEN RESERVED_LIMIT = LENGTH(V(NT-1));                         |   4664 INITIALIZA
      ELSE RESERVED_LIMIT = LENGTH(V(NT));                                         |   4718 INITIALIZA
      V(EOFILE) = 'EOF';                                                           |   4760 INITIALIZA
      STOPIT(EOFILE) = TRUE;                                                       |   4776 INITIALIZA
      CHARTYPE(BYTE(' ')) = 1;                                                     |   4786 INITIALIZA
      DO I = 0 TO 255;                                                             |   4800 INITIALIZA
        NOT_LETTER_OR_DIGIT(I) = TRUE;                                             |   4834 INITIALIZA
      END;                                                                         |   4846 INITIALIZA
      DO I = 0 TO LENGTH(ALPHABET) - 1;                                            |   4854 INITIALIZA
        J = BYTE(ALPHABET, I);                                                     |   4916 INITIALIZA
        TX(J) = I;                                                                 |   4934 INITIALIZA
        NOT_LETTER_OR_DIGIT(J) = FALSE;                                            |   4946 INITIALIZA
        CHARTYPE(J) = 4;                                                           |   4956 INITIALIZA
      END;                                                                         |   4968 INITIALIZA
      DO I = 0 TO 9;                                                               |   4976 INITIALIZA
        J = BYTE('0123456789', I);                                                 |   5010 INITIALIZA
        NOT_LETTER_OR_DIGIT(J) = FALSE;                                            |   5026 INITIALIZA
        CHARTYPE(J) = 5;                                                           |   5036 INITIALIZA
      END;                                                                         |   5048 INITIALIZA
      DO I = V_INDEX(0) TO V_INDEX(1) - 1;                                         |   5056 INITIALIZA
        J = BYTE(V(I));                                                            |   5114 INITIALIZA
        TX(J) = I;                                                                 |   5136 INITIALIZA
        CHARTYPE(J) = 7;                                                           |   5148 INITIALIZA
      END;                                                                         |   5160 INITIALIZA
      CHARTYPE(BYTE('/')) = 6;                                                     |   5168 INITIALIZA
      /* FIRST SET UP GLOBAL VARIABLES CONTROLLING SCAN, THEN CALL IT */           |   5180 INITIALIZA
      CP = 0;  TEXT_LIMIT = -1;                                                    |   5180 INITIALIZA
      TEXT = '';                                                                   |   5194 INITIALIZA
      CONTROL(BYTE('L')) = TRUE;                                                   |   5200 INITIALIZA
      CALL SCAN;                                                                   |   5212 INITIALIZA
                                                                                   |   5216 INITIALIZA
      /* INITIALIZE THE PARSE_STACK */                                             |   5216 INITIALIZA
      SP = 1;  PARSE_STACK(SP) = EOFILE;                                           |   5216 INITIALIZA
                                                                                   |   5236 INITIALIZA
   END INITIALIZATION;                                                             |   5236 INITIALIZA
                                                                                   |   5242
                                                                                   |   5242
                                                                                   |   5242
                                                                                   |   5242
                                                                                   |   5242
                                                                                   |   5242
DUMPIT:                                                                            |   5242
   PROCEDURE;     /* DUMP OUT THE STATISTICS COLLECTED DURING THIS RUN  */         |   5242
      DOUBLE_SPACE;                                                                |   5242 DUMPIT
      /*  PUT OUT THE ENTRY COUNT FOR IMPORTANT PROCEDURES */                      |   5276 DUMPIT
                                                                                   |   5278 DUMPIT
      OUTPUT = 'STACKING DECISIONS= ' || CALLCOUNT(1);                             |   5278 DUMPIT
      OUTPUT = 'SCAN                = ' || CALLCOUNT(3);                           |   5332 DUMPIT
```

```
|      OUTPUT = 'FREE STRING AREA  = ' || FREELIMIT - FREEBASE;                          |  5386 DUMPIT
|   END DUMPIT;                                                                          |  5436 DUMPIT
|                                                                                        |  5442
|                                                                                        |  5442
|STACK_DUMP:                                                                             |  5442
|   PROCEDURE;                                                                           |  5442
|      DECLARE LINE CHARACTER;                                                           |  5442 STACK_DUM
|      LINE = 'PARTIAL PARSE TO THIS POINT IS: ';                                        |  5454 STACK_DUM
|      DO I = 2 TO SP;                                                                   |  5462 STACK_DUM
|         IF LENGTH(LINE) > 105 THEN                                                     |  5506 STACK_DUM
|            DO;                                                                         |  5548 STACK_DUM
|               OUTPUT = LINE;                                                           |  5540 STACK_DUM
|               LINE = X4;                                                               |  5560 STACK_DUM
|            END;                                                                        |  5568 STACK_DUM
|         LINE = LINE || X1 || V(PARSE_STACK(I));                                        |  5568 STACK_DUM
|      END;                                                                              |  5622 STACK_DUM
|      OUTPUT = LINE;                                                                    |  5630 STACK_DUM
|   END STACK_DUMP;                                                                      |  5650 STACK_DUM
|                                                                                        |  5656
|                                                                                        |  5656
| SET_BIT:PROCEDURE(LOC);                                                 /*NEWQUAD*/    |  5656
|                                                                                        |  5656 SET_BIT
| /*SET_BIT  SET BIT AT LOCATION LOC OF MICROWORD  */                                    |  5656 SET_BIT
|                                                                                        |  5656 SET_BIT
|  DECLARE LOC FIXED;                                                     /*NEWQUAD*/    |  5656 SET_BIT
|  BYTE(BUF_MIC,LOC)="F1";                                                /*NEWQUAD*/    |  5668 SET_BIT
|  END;                                                                   /*NEWQUAD*/    |  5684 SET_BIT
| SET_FIELD:PROCEDURE(VALUE,LOC1,LOC2);                                   /*NEWQUAD*/    |  5690
|                                                                                        |  5690 SET_FIEL
| /*SET_FIELD  SET BITS AT LOCATION LOC1 TO LOC2 TO THE GIVEN VALUE  */                  |  5690 SET_FIEL
|                                                                                        |  5690 SET_FIEL
|  DECLARE (VALUE,LOC1,LOC2,A,B) FIXED;                                   /*NEWQUAD*/    |  5690 SET_FIEL
|  A=0;                                                                   /*NEWQUAD*/    |  5702 SET_FIEL
|  DO B=LOC1 TO LOC2;                                                     /*NEWQUAD*/    |  5708 SET_FIEL
|     IF VALUE MOD 2 = 1 THEN DO;                                         /*NEWQUAD*/    |  5752 SET_FIEL
|        BYTE(BUF_MIC,LOC2-A)="F1";                                       /*NEWQUAD*/    |  5778 SET_FIEL
|        END;                                                             /*NEWQUAD*/    |  5798 SET_FIEL
|     A=A+1;                                                              /*NEWQUAD*/    |  5798 SET_FIEL
|     VALUE=VALUE/2;                                                      /*NEWQUAD*/    |  5810 SET_FIEL
|     END;                                                                /*NEWQUAD*/    |  5826 SET_FIEL
|  END;                                                                   /*NEWQUAD*/    |  5834 SET_FIEL
| PUT_MIC:PROCEDURE(LOCATION);                                            /*NEWQUAD*/    |  5840
|                                                                                        |  5840 PUT_MIC
| /*PUT_MIC  PRINT OUT MICROWORD AND REINITIALIZE IT TO ALL ZERO */                      |  5840 PUT_MIC
|                                                                                        |  5840 PUT_MIC
|  DECLARE BLK(16) FIXED INITIAL                                          /*NEWQUAD*/    |  5840 PUT_MIC
|  (3,5,8,10,12,14,18,23,27,33,38,43,46,48,54,59,64);                     /*NEWQUAD*/    |  5852 PUT_MIC
|  DECLARE (LOCATION,A,B) FIXED;                                          /*NEWQUAD*/    |  5852 PUT_MIC
|  DECLARE ADDRESS CHARACTER;                                             /*NEWQUAD*/    |  5852 PUT_MIC
|  ADDRESS='  ';                                                          /*NEWQUAD*/    |  5852 PUT_MIC
|  DO A=1 TO 3;                                                           /*NEWQUAD*/    |  5860 PUT_MIC
|     B=LOCATION MOD 8;                                                   /*NEWQUAD*/    |  5896 PUT_MIC
|     ADDRESS=B||ADDRESS;                                                 /*NEWQUAD*/    |  5914 PUT_MIC
|     LOCATION=LOCATION/8;                                                /*NEWQUAD*/    |  5946 PUT_MIC
|     END;                                                                /*NEWQUAD*/    |  5962 PUT_MIC
|  OUTPUT(3)='  '||ADDRESS||BUF_MIC;                                      /*NEWQUAD*/    |  5970 PUT_MIC
|  DO A=0 TO 72;                                                          /*NEWQUAD*/    |  6044 PUT_MIC
|     BYTE(BUF_MIC,A)="F0";                                               /*NEWQUAD*/    |  6078 PUT_MIC
|     END;                                                                /*NEWQUAD*/    |  6094 PUT_MIC
|  DO A=0 TO 16;                                                          /*NEWQUAD*/    |  6102 PUT_MIC
|     BYTE(BUF_MIC,BLK(A))="40";                                          /*NEWQUAD*/    |  6136 PUT_MIC
|     END;                                                                /*NEWQUAD*/    |  6160 PUT_MIC
|  END;                                                                   /*NEWQUAD*/    |  6168 PUT_MIC
| OUT_MIC:PROCEDURE;                                                      /*NEWQUAD*/    |  6174
|                                                                                        |  6174 OUT_MIC
| /*OUT_MIC  SET NEXT ADDRESS FIELD OF MICROWORD */                                      |  6174 OUT_MIC
|                                                                          |         6174 OUT_MIC
|  IF BR_FLAG=TRUE & NEWQUAD(II+4)=BR THEN DO;              /*NEWQUAD*/ |  6174 OUT_MIC
|     LL=NEWQUAD(II+7);                                     /*NEWQUAD*/ |  6262 OUT_MIC
|     CALL SET_FIELD(MIC_LOC(LL),65,72);                    /*NEWQUAD*/ |  6282 OUT_MIC
|     END;                                                  /*NEWQUAD*/ |  6322 OUT_MIC
|  ELSE                                                     /*NEWQUAD*/ |  6322 OUT_MIC
|     CALL SET_FIELD(CURR_MIC+1,65,72);                     /*NEWQUAD*/ |  6322 OUT_MIC
|  BR_FLAG=FALSE;                                           /*NEWQUAD*/ |  6366 OUT_MIC
|  CALL PUT_MIC(CURR_MIC);                                  /*NEWQUAD*/ |  6372 OUT_MIC
|  CURR_MIC=CURR_MIC+1;                                     /*NEWQUAD*/ |  6388 OUT_MIC
|  END;                                                     /*NEWQUAD*/ |  6400 OUT_MIC
| RO_PLUS2_READ:PROCEDURE;                                  /*NEWQUAD*/ |  6406
|                                                                       |  6406 RO_PLUS2_REA
| /*RO_PLUS2_READ   GENERATE A MICROWORD TO INITIATE AREAD OPERATION    |  6406 RO_PLUS2_REA
|                   AND UODATE RO BY 2    */                            |  6406 RO_PLUS2_REA
|                                                                       |  6406 RO_PLUS2_REA
|  CALL SET_BIT(0);                                         /*NEWQUAD*/ |  6406 RO_PLUS2_REA
|  CALL SET_BIT(1);                                         /*NEWQUAD*/ |  6432 RO_PLUS2_REA
|  CALL SET_BIT(2);                                         /*NEWQUAD*/ |  6448 RO_PLUS2_REA
|  CALL SET_BIT(6);                                         /*NEWQUAD*/ |  6464 RO_PLUS2_REA
|  CALL SET_BIT(7);                                         /*NEWQUAD*/ |  6480 RO_PLUS2_REA
|  CALL SET_BIT(11);                                        /*NEWQUAD*/ |  6496 RO_PLUS2_REA
|  CALL SET_BIT(13);                                        /*NEWQUAD*/ |  6512 RO_PLUS2_REA
|  CALL SET_BIT(17);                                        /*NEWQUAD*/ |  6528 RO_PLUS2_REA
|  CALL SET_BIT(29);                                        /*NEWQUAD*/ |  6544 RO_PLUS2_REA
|  CALL SET_BIT(32);                                        /*NEWQUAD*/ |  6560 RO_PLUS2_REA
|  CALL SET_BIT(36);                                        /*NEWQUAD*/ |  6576 RO_PLUS2_REA
|  CALL SET_BIT(39);                                        /*NEWQUAD*/ |  6592 RO_PLUS2_REA
|  CALL SET_BIT(40);                                        /*NEWQUAD*/ |  6608 RO_PLUS2_REA
|  CALL SET_BIT(41);                                        /*NEWQUAD*/ |  6624 RO_PLUS2_REA
|  CALL SET_BIT(42);                                        /*NEWQUAD*/ |  6640 RO_PLUS2_REA
|  CALL SET_BIT(44);                                        /*NEWQUAD*/ |  6656 RO_PLUS2_REA
|  CALL SET_BIT(47);                                        /*NEWQUAD*/ |  6672 RO_PLUS2_REA
|  CALL SET_BIT(58);                                        /*NEWQUAD*/ |  6688 RO_PLUS2_REA
|  CALL OUT_MIC;                                            /*NEWQUAD*/ |  6704 RO_PLUS2_REA
|  END;                                                     /*NEWQUAD*/ |  6712 RO_PLUS2_REA
| R3_UNIBUS:PROCEDURE;                                      /*NEWQUAD*/ |  6718
|                                                                       |  6718 R3_UNIBUS
| /*R3_UNIBUS   GENERATE A MICROWORD TO CLOCK UNIBUS RESULT REGISTER */  |  6718 R3_UNIBUS
|                                                                       |  6718 R3_UNIBUS
|  CALL SET_BIT(1);                                         /*NEWQUAD*/ |  6718 R3_UNIBUS
|  CALL SET_BIT(6);                                         /*NEWQUAD*/ |  6746 R3_UNIBUS
|  CALL SET_BIT(7);                                         /*NEWQUAD*/ |  6762 R3_UNIBUS
|  CALL SET_BIT(45);                                        /*NEWQUAD*/ |  6778 R3_UNIBUS
|  CALL SET_BIT(58);                                        /*NEWQUAD*/ |  6794 R3_UNIBUS
|  CALL SET_FIELD(NEWQUAD(II+3),60,63);                     /*NEWQUAD*/ |  6810 R3_UNIBUS
|  CALL OUT_MIC;                                            /*NEWQUAD*/ |  6854 R3_UNIBUS
|  END;                                                     /*NEWQUAD*/ |  6862 R3_UNIBUS
| RTEMP_UNIBUS:PROCEDURE;                                   /*NEWQUAD*/ |  6868
|                                                                       |  6868 RTEMP_UNIBUS
| /*RTEMP_UNIBUS  GENERATE A MICROWORD TO CLOCK UNIBUS TO RTEMP  */     |  6868 RTEMP_UNIBUS
|                                                                       |  6868 RTEMP_UNIBUS
|  CALL SET_BIT(1);                                         /*NEWQUAD*/ |  6868 RTEMP_UNIBUS
|  CALL SET_BIT(6);                                         /*NEWQUAD*/ |  6896 RTEMP_UNIBUS
|  CALL SET_BIT(7);                                         /*NEWQUAD*/ |  6912 RTEMP_UNIBUS
|  CALL SET_BIT(45);                                        /*NEWQUAD*/ |  6928 RTEMP_UNIBUS
|  CALL SET_BIT(58);                                        /*NEWQUAD*/ |  6944 RTEMP_UNIBUS
|  CALL SET_BIT(60);                                        /*NEWQUAD*/ |  6960 RTEMP_UNIBUS
|  CALL OUT_MIC;                                            /*NEWQUAD*/ |  6976 RTEMP_UNIBUS
|  END;                                                     /*NEWQUAD*/ |  6984 RTEMP_UNIBUS
| BA_RO_READ:PROCEDURE;                                     /*NEWQUAD*/ |  6990
|                                                                       |  6990 BA_RO_READ
|  /* BA_RO_READ  GENERATE A MICROWORD TO CLOCK RO TO BA REGISTER       |  6990 BA_RO_READ
|                 AND INITIATE A READ OPERATION   */                    |  6990 BA_RO_READ
|                                                                       |  6990 BA_RO_READ
|  CALL SET_BIT(1);                                         /*NEWQUAD*/ |  6990 BA_RO_READ
|  CALL SET_BIT(2);                                                /*NEWQUAD*/ |  7018 BA_RO_READ
|  CALL SET_BIT(13);                                               /*NEWQUAD*/ |  7034 BA_RO_READ
|  CALL SET_BIT(17);                                               /*NEWQUAD*/ |  7050 BA_RO_READ
|  CALL SET_BIT(47);                                               /*NEWQUAD*/ |  7066 BA_RO_READ
|  CALL SET_BIT(58);                                               /*NEWQUAD*/ |  7082 BA_RO_READ
|  CALL OUT_MIC;                                                   /*NEWQUAD*/ |  7098 BA_RO_READ
|  END;                                                            /*NEWQUAD*/ |  7106 BA_RO_READ
| R3_D:PROCEDURE;                                                  /*NEWQUAD*/ |  7112
|                                                                              |  7112 R3_D
| /*R3_D  GENERATE A MICROWORD TO CLOCK D REGISTER TO RESULT REGISTER   */     |  7112 R3_D
|                                                                              |  7112 R3_D
|  CALL SET_BIT(1);                                                /*NEWQUAD*/ |  7112 R3_D
|  CALL SET_BIT(6);                                                /*NEWQUAD*/ |  7140 R3_D
|  CALL SET_BIT(7);                                                /*NEWQUAD*/ |  7156 R3_D
|  CALL SET_BIT(44);                                               /*NEWQUAD*/ |  7172 R3_D
|  CALL SET_BIT(58);                                               /*NEWQUAD*/ |  7188 R3_D
|  CALL SET_FIELD(NEWQUAD(II+3),60,63);                            /*NEWQUAD*/ |  7204 R3_D
|  CALL OUT_MIC;                                                   /*NEWQUAD*/ |  7248 R3_D
|  END;                                                            /*NEWQUAD*/ |  7256 R3_D
| B_R2:PROCEDURE;                                                  /*NEWQUAD*/ |  7262
|                                                                              |  7262 B_R2
| /*B_R2  GENERATE A MICRO WORD TO CLOCK OPERAND2 REGISTER TO B REGISTER */    |  7262 B_R2
|                                                                              |  7262 B_R2
|  CALL SET_BIT(1);                                                /*NEWQUAD*/ |  7262 B_R2
|  CALL SET_BIT(9);                                                /*NEWQUAD*/ |  7290 B_R2
|  CALL SET_BIT(58);                                               /*NEWQUAD*/ |  7306 B_R2
|  CALL SET_FIELD(NEWQUAD(II+2),60,63);                            /*NEWQUAD*/ |  7322 B_R2
|  CALL OUT_MIC;                                                   /*NEWQUAD*/ |  7366 B_R2
|  END;                                                            /*NEWQUAD*/ |  7374 B_R2
| RTEMP_4:PROCEDURE;                                               /*NEWQUAD*/ |  7380
|                                                                              |  7380 RTEMP_4
| /*RTEMP_4   GENERATE A MICROWORD TO SET RTEMP TO 4    */                     |  7380 RTEMP_4
|                                                                              |  7380 RTEMP_4
|  CALL SET_BIT(0);                                                /*NEWQUAD*/ |  7380 RTEMP_4
|  CALL SET_BIT(1);                                                /*NEWQUAD*/ |  7406 RTEMP_4
|  CALL SET_BIT(6);                                                /*NEWQUAD*/ |  7422 RTEMP_4
|  CALL SET_BIT(7);                                                /*NEWQUAD*/ |  7438 RTEMP_4
|  CALL SET_BIT(11);                                               /*NEWQUAD*/ |  7454 RTEMP_4
|  CALL SET_BIT(28);                                               /*NEWQUAD*/ |  7470 RTEMP_4
|  CALL SET_BIT(29);                                               /*NEWQUAD*/ |  7486 RTEMP_4
|  CALL SET_BIT(31);                                               /*NEWQUAD*/ |  7502 RTEMP_4
|  CALL SET_BIT(34);                                               /*NEWQUAD*/ |  7518 RTEMP_4
|  CALL SET_BIT(35);                                               /*NEWQUAD*/ |  7534 RTEMP_4
|  CALL SET_BIT(36);                                               /*NEWQUAD*/ |  7550 RTEMP_4
|  CALL SET_BIT(37);                                               /*NEWQUAD*/ |  7566 RTEMP_4
|  CALL SET_BIT(39);                                               /*NEWQUAD*/ |  7582 RTEMP_4
|  CALL SET_BIT(40);                                               /*NEWQUAD*/ |  7598 RTEMP_4
|  CALL SET_BIT(41);                                               /*NEWQUAD*/ |  7614 RTEMP_4
|  CALL SET_BIT(42);                                               /*NEWQUAD*/ |  7630 RTEMP_4
|  CALL SET_BIT(44);                                               /*NEWQUAD*/ |  7646 RTEMP_4
|  CALL SET_BIT(58);                                               /*NEWQUAD*/ |  7662 RTEMP_4
|  CALL SET_BIT(60);                                               /*NEWQUAD*/ |  7678 RTEMP_4
|  CALL OUT_MIC;                                                   /*NEWQUAD*/ |  7694 RTEMP_4
|  END;                                                            /*NEWQUAD*/ |  7702 RTEMP_4
|  R3_O:PROCEDURE;                                                 /*NEWQUAD*/ |  7708
|                                                                              |  7708 R3_O
| /*R3_O   GENERATE A MICROWORD TO SET RESULT REGISTER TO ZERO   */            |  7708 R3_O
|                                                                              |  7708 R3_O
|  CALL SET_BIT(0);                                                /*NEWQUAD*/ |  7708 R3_O
|  CALL SET_BIT(1);                                                /*NEWQUAD*/ |  7734 R3_O
|  CALL SET_BIT(6);                                                /*NEWQUAD*/ |  7750 R3_O
|  CALL SET_BIT(7);                                                /*NEWQUAD*/ |  7766 R3_O
|  CALL SET_BIT(11);                                               /*NEWQUAD*/ |  7782 R3_O
|  CALL SET_BIT(31);                                               /*NEWQUAD*/ |  7798 R3_O
|  CALL SET_BIT(32);                                               /*NEWQUAD*/ |  7814 R3_O
|  CALL SET_BIT(44);                                               /*NEWQUAD*/ |  7830 R3_O
```

```
 | CALL SET_BIT(53);                                                     /*NEWQUAD*/ |  7846 R3_0
 | CALL SET_FIELD(NEWQUAD(II+3),60,63);                                  /*NEWQUAD*/ |  7862 R3_0
 | CALL OUT_MIC;                                                         /*NEWQUAD*/ |  7906 R3_0
 | END;                                                                  /*NEWQUAD*/ |  7914 R3_0
 | RTEMP_SHIFT:PROCEDURE;                                                /*NEWQUAD*/ |  7920
 |                                                                                   |  7920 RTEMP_SHIF
 | /*RTEMP_SHIFT  GENERATE A MICROWORD TO SHIFT RTEMP ONE POSITION TO LEFT */        |  7920 RTEMP_SHIF
 |                                                                                   |  7920 RTEMP_SHIF
 |  CALL SET_BIT(0);                                                     /*NEWQUAD*/ |  7920 RTEMP_SHIF
 |  CALL SET_BIT(1);                                                     /*NEWQUAD*/ |  7946 RTEMP_SHIF
 |  CALL SET_BIT(6);                                                     /*NEWQUAD*/ |  7962 RTEMP_SHIF
 |  CALL SET_BIT(7);                                                     /*NEWQUAD*/ |  7978 RTEMP_SHIF
 |  CALL SET_BIT(11);                                                    /*NEWQUAD*/ |  7994 RTEMP_SHIF
 |  CALL SET_BIT(29);                                                    /*NEWQUAD*/ |  8010 RTEMP_SHIF
 |  CALL SET_BIT(30);                                                    /*NEWQUAD*/ |  8026 RTEMP_SHIF
 |  CALL SET_BIT(44);                                                    /*NEWQUAD*/ |  8042 RTEMP_SHIF
 |  CALL SET_BIT(58);                                                    /*NEWQUAD*/ |  8058 RTEMP_SHIF
 |  CALL SET_BIT(60);                                                    /*NEWQUAD*/ |  8074 RTEMP_SHIF
 |  CALL OUT_MIC;                                                        /*NEWQUAD*/ |  8090 RTEMP_SHIFT
 |  END;                                                                 /*NEWQUAD*/ |  8098 RTEMP_SHIFT
 | B_R1:PROCEDURE;                                                       /*NEWQUAD*/ |  8104
 |                                                                                   |  8104 B_R1
 | /*B_R1  GENERATE A MICROWORD TO SET B REGISTER T1 REGISTER   */                   |  8104 B_R1
 |                                                                                   |  8104 B_R1
 |  CALL SET_BIT(1);                                                     /*NEWQUAD*/ |  8104 B_R1
 |  CALL SET_BIT(9);                                                     /*NEWQUAD*/ |  8132 B_R1
 |  CALL SET_BIT(58);                                                    /*NEWQUAD*/ |  8148 B_R1
 |  CALL SET_FIELD(NEWQUAD(II+1),60,63);                                 /*NEWQUAD*/ |  8164 B_R1
 |  END;                                                                 /*NEWQUAD*/ |  8208 B_R1
 | R2_SHIFT:PROCEDURE;                                                   /*NEWQUAD*/ |  8214
 |                                                                                   |  8214 R2_SHIFT
 | /*R2_SHIFT  GENERATE A MICROWORD TO SHIFT OPERAND2 REGISTER TO LEFT               |  8214 R2_SHIFT
 |             ONE POSITION      */                                                  |  8214 R2_SHIFT
 |                                                                                   |  8214 R2_SHIFT
 |  CALL SET_BIT(0);                                                     /*NEWQUAD*/ |  8214 R2_SHIFT
 |  CALL SET_BIT(1);                                                     /*NEWQUAD*/ |  8240 R2_SHIFT
 |  CALL SET_BIT(1);                                                     /*NEWQUAD*/ |  8256 R2_SHIFT
 |  CALL SET_BIT(6);                                                     /*NEWQUAD*/ |  8272 R2_SHIFT
 |  CALL SET_BIT(7);                                                     /*NEWQUAD*/ |  8288 R2_SHIFT
 |  CALL SET_BIT(11);                                                    /*NEWQUAD*/ |  8304 R2_SHIFT
 |  CALL SET_BIT(24);                                                    /*NEWQUAD*/ |  8320 R2_SHIFT
 |  CALL SET_BIT(44);                                                    /*NEWQUAD*/ |  8336 R2_SHIFT
 |  CALL SET_BIT(49);                                                    /*NEWQUAD*/ |  8352 R2_SHIFT
 |  CALL SET_BIT(52);                                                    /*NEWQUAD*/ |  8368 R2_SHIFT
 |  CALL SET_BIT(53);                                                    /*NEWQUAD*/ |  8384 R2_SHIFT
 |  CALL SET_BIT(58);                                                    /*NEWQUAD*/ |  8400 R2_SHIFT
 |  CALL SET_FIELD(NEWQUAD(II+2),60,63);                                 /*NEWQUAD*/ |  8416 R2_SHIFT
 |  CALL OUT_MIC;                                                        /*NEWQUAD*/ |  8460 R2_SHIFT
 |  END;                                                                 /*NEWQUAD*/ |  8468 R2_SHIFT
 | NO_OP:PROCEDURE;                                                      /*NEWQUAD*/ |  8474
 |                                                                                   |  8474 NO_OP
 | /*NO_OP  GENERATE A NO_OP MICROWORD    */                                         |  8474 NO_OP
 |                                                                                   |  8474 NO_OP
 |  CALL SET_BIT(1);                                                     /*NEWQUAD*/ |  8474 NO_OP
 |  CALL OUT_MIC;                                                        /*NEWQUAD*/ |  8502 NO_OP
 |  END;                                                                 /*NEWQUAD*/ |  8510 NO_OP
 | R3_R3_PLUS_B:PROCEDURE;                                               /*NEWQUAD*/ |  8516
 |                                                                                   |  8516 R3_R3_PLUS_
 | /*R3_R3_PLUS_B  GENERATE A MICROWORD TO ADD RESULT REGISTER TO B REGISTER         |  8516 R3_R3_PLUS_
 |                 AND PUT RESULT IN RESULT REGISTER   */                            |  8516 R3_R3_PLUS_
 |                                                                                   |  8516 R3_R3_PLUS_
 |  CALL SET_BIT(0);                                                     /*NEWQUAD*/ |  8516 R3_R3_PLUS_
 |  CALL SET_BIT(6);                                                     /*NEWQUAD*/ |  8542 R3_R3_PLUS_
 |  CALL SET_BIT(7);                                                     /*NEWQUAD*/ |  8558 R3_R3_PLUS_
 |  CALL SET_BIT(11);                                                    /*NEWQUAD*/ |  8574 R3_R3_PLUS_
 |  CALL SET_BIT(29);                                                    /*NEWQUAD*/ |  8590 R3_R3_PLUS_
 |  CALL SET_BIT(32);                                        /*NEWQUAD*/ |  8606 R3_R3_PLUS_B
 |  CALL SET_BIT(58);                                        /*NEWQUAD*/ |  8622 R3_R3_PLUS_B
 |  CALL SET_FIELD(NEWQUAD(I+3),60,63);                      /*NEWQUAD*/ |  8638 R3_R3_PLUS_B
 |  CALL OUT_MIC;                                            /*NEWQUAD*/ |  8682 R3_R3_PLUS_B
 |  END;                                                     /*NEWQUAD*/ |  8690 R3_R3_PLUS_B
 | RTEMP_MINUS_1:PROCEDURE;                                  /*NEWQUAD*/ |  8696
 |                                                                       |  8696 RTEMP_MINUS_1
 | /*RTEMP_MINUS_1  GENERATE A MICROWORD TO SUBTRACT 1 FROM RTEMP     */ |  8696 RTEMP_MINUS_1
 |                                                                       |  8696 RTEMP_MINUS_1
 |  CALL SET_BIT(0);                                         /*NEWQUAD*/ |  8696 RTEMP_MINUS_1
 |  CALL SET_BIT(6);                                         /*NEWQUAD*/ |  8722 RTEMP_MINUS_1
 |  CALL SET_BIT(7);                                         /*NEWQUAD*/ |  8738 RTEMP_MINUS_1
 |  CALL SET_BIT(11);                                        /*NEWQUAD*/ |  8754 RTEMP_MINUS_1
 |  CALL SET_BIT(26);                                        /*NEWQUAD*/ |  8770 RTEMP_MINUS_1
 |  CALL SET_BIT(30);                                        /*NEWQUAD*/ |  8786 RTEMP_MINUS_1
 |  CALL SET_BIT(31);                                        /*NEWQUAD*/ |  8802 RTEMP_MINUS_1
 |  CALL SET_BIT(37);                                        /*NEWQUAD*/ |  8818 RTEMP_MINUS_1
 |  CALL SET_BIT(39);                                        /*NEWQUAD*/ |  8834 RTEMP_MINUS_1
 |  CALL SET_BIT(40);                                        /*NEWQUAD*/ |  8850 RTEMP_MINUS_1
 |  CALL SET_BIT(41);                                        /*NEWQUAD*/ |  8866 RTEMP_MINUS_1
 |  CALL SET_BIT(42);                                        /*NEWQUAD*/ |  8882 RTEMP_MINUS_1
 |  CALL SET_BIT(44);                                        /*NEWQUAD*/ |  8898 RTEMP_MINUS_1
 |  CALL SET_BIT(50);                                        /*NEWQUAD*/ |  8914 RTEMP_MINUS_1
 |  CALL SET_BIT(52);                                        /*NEWQUAD*/ |  8930 RTEMP_MINUS_1
 |  CALL SET_BIT(58);                                        /*NEWQUAD*/ |  8946 RTEMP_MINUS_1
 |  CALL SET_BIT(60);                                        /*NEWQUAD*/ |  8962 RTEMP_MINUS_1
 |  END;                                                     /*NEWQUAD*/ |  8978 RTEMP_MINUS_1
 | MIC_GEN:PROCEDURE;                                        /*NEWQUAD*/ |  8984
 |  DECLARE CASE_NUM(30) FIXED INITIAL (9,                   /*NEWQUAD*/ |  8984 MIC_GEN
 |                                      0,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                     11,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      1,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      0,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      0,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      0,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                     10,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      2,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      0,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      0,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      1,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      1,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      1,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      0,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      0,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      0,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      0,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      0,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      0,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      0,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      0,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      0,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      0,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      8,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      4,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      5,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      6,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      7,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      3,                   /*NEWQUAD*/ |  8996 MIC_GEN
 |                                      0);                  /*NEWQUAD*/ |  8996 MIC_GEN
 |  DECLARE TEMP_CURR_MIC FIXED;                             /*NEWQUAD*/ |  8996 MIC_GEN
 |  DO CASE CASE_NUM(OPER);                                  /*NEWQUAD*/ |  8996 MIC_GEN
 |     DO;                                                   /*NEWQUAD*/ |  9028 MIC_GEN
 |                                                                       |  9028 MIC_GEN CASE
 | /*GENERATE MICROCODE FOR ADD REGISTER QUAD  */                        |  9028 MIC_GEN
 |                                                                       |  9028 MIC_GEN
 |        CALL B_32;                                         /*NEWQUAD*/ |  9028 MIC_GEN
```

```
            CALL SET_BIT(0);                                      /*NEWQUAD*/ |  9034 MIC_GEN
            CALL SET_BIT(11);                                     /*NEWQUAD*/ |  9050 MIC_GEN
            CALL SET_BIT(29);                                     /*NEWQUAD*/ |  9066 MIC_GEN
            CALL SET_BIT(32);                                     /*NEWQUAD*/ |  9082 MIC_GEN
            CALL SET_BIT(58);                                     /*NEWQUAD*/ |  9098 MIC_GEN
            CALL SET_FIELD(NEWQUAD(II+1),60,63);                  /*NEWQUAD*/ |  9114 MIC_GEN
            CALL OUT_MIC;                                         /*NEWQUAD*/ |  9158 MIC_GEN
            BR_FLAG=TRUE;                                         /*NEWQUAD*/ |  9166 MIC_GEN
            CALL R3_D;                                            /*NEWQUAD*/ |  9174 MIC_GEN
            END;                                                  /*NEWQUAD*/ |  9182 MIC_GEN
         DO;                                                      /*NEWQUAD*/ |  9182 MIC_GEN
                                                                              |  9190 MIC_GEN CASE 1
     /*GENERATE MICROCODE FOR SUB, LT, GT, EQ REGISTER QUADS    */            |  9190 MIC_GEN
                                                                              |  9190 MIC_GEN
            IF OPER=LT THEN DO;                                   /*NEWQUAD*/ |  9190 MIC_GEN
               TEMP_CURR_MIC=NEWQUAD(II+1);                       /*NEWQUAD*/ |  9206 MIC_GEN
               NEWQUAD(II+1)=NEWQUAD(II+2);                       /*NEWQUAD*/ |  9226 MIC_GEN
               NEWQUAD(II+2)=TEMP_CURR_MIC;                       /*NEWQUAD*/ |  9258 MIC_GEN
               END;                                               /*NEWQUAD*/ |  9278 MIC_GEN
            CALL B_R2;                                            /*NEWQUAD*/ |  9278 MIC_GEN
            CALL SET_BIT(0);                                      /*NEWQUAD*/ |  9286 MIC_GEN
            CALL SET_BIT(11);                                     /*NEWQUAD*/ |  9300 MIC_GEN
            CALL SET_BIT(19);                                     /*NEWQUAD*/ |  9316 MIC_GEN
            CALL SET_BIT(25);                                     /*NEWQUAD*/ |  9332 MIC_GEN
            CALL SET_BIT(26);                                     /*NEWQUAD*/ |  9348 MIC_GEN
            CALL SET_BIT(30);                                     /*NEWQUAD*/ |  9364 MIC_GEN
            CALL SET_BIT(31);                                     /*NEWQUAD*/ |  9380 MIC_GEN
            CALL SET_BIT(58);                                     /*NEWQUAD*/ |  9396 MIC_GEN
            CALL SET_FIELD(NEWQUAD(II+1),60,63);                  /*NEWQUAD*/ |  9412 MIC_GEN
            CALL OUT_MIC;                                         /*NEWQUAD*/ |  9456 MIC_GEN
            BR_FLAG=TRUE;                                         /*NEWQUAD*/ |  9464 MIC_GEN
            CALL R3_D;                                            /*NEWQUAD*/ |  9472 MIC_GEN
            END;                                                  /*NEWQUAD*/ |  9480 MIC_GEN
         DO;                                                      /*NEWQUAD*/ |  9480 MIC_GEN
                                                                              |  9488 MIC_GEN CASE 2.
     /*GENERATE MICROCODE FOR BT REGISTER QUADS    */                         |  9488 MIC_GEN
                                                                              |  9488 MIC_GEN
            CALL SET_BIT(0);                                      /*NEWQUAD*/ |  9488 MIC_GEN
            CALL SET_BIT(11);                                     /*NEWQUAD*/ |  9502 MIC_GEN
            CALL SET_BIT(26);                                     /*NEWQUAD*/ |  9518 MIC_GEN
            CALL SET_BIT(28);                                     /*NEWQUAD*/ |  9534 MIC_GEN
            CALL SET_BIT(31);                                     /*NEWQUAD*/ |  9550 MIC_GEN
            CALL SET_BIT(32);                                     /*NEWQUAD*/ |  9566 MIC_GEN
            CALL SET_BIT(37);                                     /*NEWQUAD*/ |  9582 MIC_GEN
            CALL SET_BIT(39);                                     /*NEWQUAD*/ |  9598 MIC_GEN
            CALL SET_BIT(40);                                     /*NEWQUAD*/ |  9614 MIC_GEN
            CALL SET_BIT(41);                                     /*NEWQUAD*/ |  9630 MIC_GEN
            CALL SET_BIT(42);                                     /*NEWQUAD*/ |  9646 MIC_GEN
            CALL SET_BIT(50);                                     /*NEWQUAD*/ |  9662 MIC_GEN
            CALL SET_BIT(52);                                     /*NEWQUAD*/ |  9678 MIC_GEN
            TEMP_CURR_MIC=CURR_MIC;                               /*NEWQUAD*/ |  9694 MIC_GEN
            IF CURR_MIC MOD 2 = 1 THEN                            /*NEWQUAD*/ |  9702 MIC_GEN
               CURR_MIC=CURR_MIC+1;                               /*NEWQUAD*/ |  9736 MIC_GEN
            CALL SET_FIELD(CURR_MIC+1,65,72);                     /*NEWQUAD*/ |  9740 MIC_GEN
            CALL PUT_MIC(TEMP_CURR_MIC);                          /*NEWQUAD*/ |  9776 MIC_GEN
            CURR_MIC=CURR_MIC+1;                                  /*NEWQUAD*/ |  9792 MIC_GEN
            CALL SET_BIT(1);                                      /*NEWQUAD*/ |  9804 MIC_GEN
            CALL OUT_MIC;                                         /*NEWQUAD*/ |  9820 MIC_GEN
            CALL SET_BIT(1);                                      /*NEWQUAD*/ |  9828 MIC_GEN
            CALL SET_FIELD(CURR_MIC+2,65,72);                     /*NEWQUAD*/ |  9844 MIC_GEN
            CALL PUT_MIC(CURR_MIC);                               /*NEWQUAD*/ |  9880 MIC_GEN
            CURR_MIC=CURR_MIC+1;                                  /*NEWQUAD*/ |  9896 MIC_GEN
            CALL SET_BIT(1);                                      /*NEWQUAD*/ |  9908 MIC_GEN
            LL=NEWQUAD(II+3);                                     /*NEWQUAD*/ |  9924 MIC_GEN
            CALL SET_FIELD(MIC_LOC(LL),65,72);                    /*NEWQUAD*/ |  9944 MIC_GEN
            CALL PUT_MIC(CURR_MIC);                               /*NEWQUAD*/ |  9984 MIC_GEN
```

```
          CURR_MIC=CURR_MIC+1;                               /*NEWQUAD*/ | 10012 MIC_GEN
          END;                                               /*NEWQUAD*/ | 10012 MIC_GEN
       DO;                                                   /*NEWQUAD*/ | 10012 MIC_GEN
                                                                         | 10020 MIC_GEN CASE 3
  /*GENERATE MICROCODE FOR ASL REGISTER QUADS     */                     | 10020 MIC_GEN
                                                                         | 10020 MIC_GEN
          CALL SET_BIT(0);                                   /*NEWQUAD*/ | 10020 MIC_GEN
          CALL SET_BIT(11);                                  /*NEWQUAD*/ | 10034 MIC_GEN
          CALL SET_BIT(29);                                  /*NEWQUAD*/ | 10050 MIC_GEN
          CALL SET_BIT(30);                                  /*NEWQUAD*/ | 10066 MIC_GEN
          CALL SET_BIT(58);                                  /*NEWQUAD*/ | 10082 MIC_GEN
          CALL SET_FIELD(NEWQUAD(II+1),60,63);               /*NEWQUAD*/ | 10098 MIC_GEN
          CALL OUT_MIC;                                      /*NEWQUAD*/ | 10142 MIC_GEN
          CALL R3_D;                                         /*NEWQUAD*/ | 10150 MIC_GEN
          END;                                               /*NEWQUAD*/ | 10158 MIC_GEN
       DO;                                                   /*NEWQUAD*/ | 10158 MIC_GEN
                                                                         | 10166 MIC_GEN CASE 4
  /*GENERATE MICROCODE FOR RD REGISTER QUADS*/                           | 10166 MIC_GEN
                                                                         | 10166 MIC_GEN
          CALL RO_PLUS2_READ;                                /*NEWQUAD*/ | 10166 MIC_GEN
          CALL RTEMP_UNIBUS;                                 /*NEWQUAD*/ | 10174 MIC_GEN
          CALL SET_BIT(60);                                  /*NEWQUAD*/ | 10182 MIC_GEN
          CALL BA_RO_READ;                                   /*NEWQUAD*/ | 10198 MIC_GEN
          BR_FLAG=TRUE;                                      /*NEWQUAD*/ | 10206 MIC_GEN
          CALL R3_UNIBUS;                                    /*NEWQUAD*/ | 10214 MIC_GEN
          END;                                               /*NEWQUAD*/ | 10222 MIC_GEN
       DO;                                                   /*NEWQUAD*/ | 10222 MIC_GEN
                                                                         | 10230 MIC_GEN CASE 5
  /*GENERATE MICROCODE FOR WT REGISTER QUADS      */                     | 10230 MIC_GEN
                                                                         | 10230 MIC_GEN
          CALL RO_PLUS2_READ;                                /*NEWQUAD*/ | 10230 MIC_GEN
          CALL RTEMP_UNIBUS;                                 /*NEWQUAD*/ | 10238 MIC_GEN
          CALL SET_BIT(1);                                   /*NEWQUAD*/ | 10246 MIC_GEN
          CALL SET_BIT(13);                                  /*NEWQUAD*/ | 10262 MIC_GEN
          CALL SET_BIT(47);                                  /*NEWQUAD*/ | 10278 MIC_GEN
          CALL SET_BIT(58);                                  /*NEWQUAD*/ | 10294 MIC_GEN
          CALL SET_BIT(60);                                  /*NEWQUAD*/ | 10310 MIC_GEN
          CALL OUT_MIC;                                      /*NEWQUAD*/ | 10326 MIC_GEN
          CALL SET_BIT(0);                                   /*NEWQUAD*/ | 10334 MIC_GEN
          CALL SET_BIT(2);                                   /*NEWQUAD*/ | 10348 MIC_GEN
          CALL SET_BIT(11);                                  /*NEWQUAD*/ | 10364 MIC_GEN
          CALL SET_BIT(15);                                  /*NEWQUAD*/ | 10380 MIC_GEN
          CALL SET_BIT(17);                                  /*NEWQUAD*/ | 10396 MIC_GEN
       CALL SET_BIT(58);                                     /*NEWQUAD*/ | 10412 MIC_GEN
          CALL SET_FIELD(NEWQUAD(II+1),60,63);               /*NEWQUAD*/ | 10428 MIC_GEN
          BR_FLAG=TRUE;                                      /*NEWQUAD*/ | 10472 MIC_GEN
          CALL OUT_MIC;                                      /*NEWQUAD*/ | 10480 MIC_GEN
          END;                                               /*NEWQUAD*/ | 10488 MIC_GEN
       DO;                                                   /*NEWQUAD*/ | 10488 MIC_GEN
                                                                         | 10496 MIC_GEN CASE 6
  /*GENERATE MICROCODE FOR RDAD REGISTER QUAD     */                     | 10496 MIC_GEN
                                                                         | 10496 MIC_GEN
          CALL RO_PLUS2_READ;                                /*NEWQUAD*/ | 10496 MIC_GEN
          CALL R3_UNIBUS;                                    /*NEWQUAD*/ | 10504 MIC_GEN
          END;                                               /*NEWQUAD*/ | 10512 MIC_GEN
       DO;                                                   /*NEWQUAD*/ | 10512 MIC_GEN
                                                                         | 10520 MIC_GEN CASE 7
  /*GENERATE MICROCODE FOR RDVR REGISTER QUAD     */                     | 10520 MIC_GEN
                                                                         | 10520 MIC_GEN
          CALL SET_FIELD(NEWQUAD(II+1),60,63);               /*NEWQUAD*/ | 10520 MIC_GEN
          CALL BA_RO_READ;                                   /*NEWQUAD*/ | 10564 MIC_GEN
          BR_FLAG=TRUE;                                      /*NEWQUAD*/ | 10572 MIC_GEN
          CALL R3_UNIBUS;                                    /*NEWQUAD*/ | 10580 MIC_GEN
          END;                                               /*NEWQUAD*/ | 10588 MIC_GEN
       DO;                                                   /*NEWQUAD*/ | 10588 MIC_GEN
                                                                         | 10596 MIC_GEN CASE 8
```

```
| /*GENERATE MICROCODE FOR LAB REGISTER QUAD    */                        | 10596 MIC_GEN
|                                                                          | 10596 MIC_GEN
|          CALL BA_RO_READ;                                   /*NEWQUAD*/ | 10596 MIC_GEN
|          CALL SET_BIT(1);                                   /*NEWQUAD*/ | 10604 MIC_GEN
|          CALL SET_BIT(6);                                   /*NEWQUAD*/ | 10620 MIC_GEN
|          CALL SET_BIT(7);                                   /*NEWQUAD*/ | 10636 MIC_GEN
|          CALL SET_BIT(45);                                  /*NEWQUAD*/ | 10652 MIC_GEN
|          CALL SET_BIT(58);                                  /*NEWQUAD*/ | 10668 MIC_GEN
|          CALL OUT_MIC;                                      /*NEWQUAD*/ | 10684 MIC_GEN
|          END;                                               /*NEWQUAD*/ | 10692 MIC_GEN
|  DO;                                                        /*NEWQUAD*/ | 10692 MIC_GEN
|                                                                          | 10700 MIC_GEN CASE 9
| /* GENERATE MICROCODE FOR WTAD REGISTER QUAD    */                       | 10700 MIC_GEN
|                                                                          | 10700 MIC_GEN
|       CALL SET_BIT(1);                                      /*NEWQUAD*/ | 10700 MIC_GEN
|       CALL SET_BIT(13);                                     /*NEWQUAD*/ | 10716 MIC_GEN
|       CALL SET_BIT(47);                                     /*NEWQUAD*/ | 10732 MIC_GEN
|       CALL SET_BIT(58);                                     /*NEWQUAD*/ | 10748 MIC_GEN
|       CALL SET_FIELD(NEWQUAD(II+1),60,63);                  /*NEWQUAD*/ | 10764 MIC_GEN
|       CALL OUT_MIC;                                         /*NEWQUAD*/ | 10808 MIC_GEN
|       CALL SET_BIT(0);                                      /*NEWQUAD*/ | 10816 MIC_GEN
|       CALL SET_BIT(2);                                      /*NEWQUAD*/ | 10830 MIC_GEN
|       CALL SET_BIT(11);                                     /*NEWQUAD*/ | 10846 MIC_GEN
|       CALL SET_BIT(15);                                     /*NEWQUAD*/ | 10862 MIC_GEN
|       CALL SET_BIT(17);                                     /*NEWQUAD*/ | 10878 MIC_GEN
|       CALL SET_BIT(58);                                     /*NEWQUAD*/ | 10894 MIC_GEN
|       CALL SET_FIELD(NEWQUAD(II+3),60,63);                  /*NEWQUAD*/ | 10910 MIC_GEN
|       BR_FLAG=TRUE;                                         /*NEWQUAD*/ | 10954 MIC_GEN
|       CALL OUT_MIC;                                         /*NEWQUAD*/ | 10962 MIC_GEN
|       END;                                                  /*NEWQUAD*/ | 10970 MIC_GEN
|                                                                          | 10970 MIC_GEN
| /*SR REGISTER QUAD*/                                                     | 10970 MIC_GEN
|                                                                          | 10970 MIC_GEN
|       ;                                                     /*NEWQUAD*/ | 10970 MIC_GEN
|       DO;                                                   /*NEWQUAD*/ | 10978 MIC_GEN CASE 1
|                                                                          | 10986 MIC_GEN CASE 1
| /* GENERATE MICROCODE FOR MUL REGISTER QUAD  */                          | 10986 MIC_GEN
|                                                                          | 10986 MIC_GEN
|          CALL RTEMP_4;                                      /*NEWQUAD*/ | 10986 MIC_GEN
|          CALL RTEMP_SHIFT;                                  /*NEWQUAD*/ | 10994 MIC_GEN
|          CALL RTEMP_SHIFT;                                  /*NEWQUAD*/ | 11002 MIC_GEN
|          CALL R3_0;                                         /*NEWQUAD*/ | 11010 MIC_GEN
|          TEMP_CURR_MIC=CURR_MIC;                            /*NEWQUAD*/ | 11018 MIC_GEN
|          IF CURR_MIC MOD 2 = 0 THEN DO;                     /*NEWQUAD*/ | 11026 MIC_GEN
|             CURR_MIC=CURR_MIC+1;                            /*NEWQUAD*/ | 11052 MIC_GEN
|             END;                                            /*NEWQUAD*/ | 11064 MIC_GEN
|          CALL B_R1;                                         /*NEWQUAD*/ | 11064 MIC_GEN
|          CALL SET_FIELD(CURR_MIC+1,65,72);                  /*NEWQUAD*/ | 11072 MIC_GEN
|          CALL PUT_MIC(TEMP_CURR_MIC);                       /*NEWQUAD*/ | 11108 MIC_GEN
|          CURR_MIC=CURR_MIC+1;                               /*NEWQUAD*/ | 11124 MIC_GEN
|          TEMP_CURR_MIC=CURR_MIC;                            /*NEWQUAD*/ | 11136 MIC_GEN
|          CALL R2_SHIFT;                                     /*NEWQUAD*/ | 11144 MIC_GEN
|          CALL NO_OP;                                        /*NEWQUAD*/ | 11152 MIC_GEN
|          CALL R3_R3_PLUS_B;                                 /*NEWQUAD*/ | 11160 MIC_GEN
|          CALL RTEMP_MINUS_1;                                /*NEWQUAD*/ | 11168 MIC_GEN
|          CALL SET_FIELD(CURR_MIC+2,65,72);                  /*NEWQUAD*/ | 11176 MIC_GEN
|          CALL PUT_MIC(CURR_MIC);                            /*NEWQUAD*/ | 11212 MIC_GEN
|          CURR_MIC=CURR_MIC+2;                               /*NEWQUAD*/ | 11228 MIC_GEN
|          CALL SET_BIT(1);                                   /*NEWQUAD*/ | 11240 MIC_GEN
|          CALL SET_FIELD(TEMP_CURR_MIC,65,72);               /*NEWQUAD*/ | 11256 MIC_GEN
|          CALL PUT_MIC(CURR_MIC);                            /*NEWQUAD*/ | 11288 MIC_GEN
|          CURR_MIC=CURR_MIC+1;                               /*NEWQUAD*/ | 11304 MIC_GEN
|          BR_FLAG=TRUE;                                      /*NEWQUAD*/ | 11316 MIC_GEN
|          CALL NO_OP;                                        /*NEWQUAD*/ | 11324 MIC_GEN
|          END;                                               /*NEWQUAD*/ | 11332 MIC_GEN
|       END;                                                  /*NEWQUAD*/ | 11332 MIC_GEN
|  END;                                                              /*NEWQUAD*/ | [illegible]
| PUT_NEWQUAD:PROCEDURE;                                             /*NEWQUAD*/ | 11346
|                                                                                | 11346 PUT_NEWQUAD
| /*PUT_NEWQUAD  PRINT OUT REGISTER QUADS  */                                    | 11346 PUT_NEWQUAD
|                                                                                | 11346 PUT_NEWQUAD
|  NEWQUAD(NEWQUADNO) = OPRTOR;                                      /*NEWQUAD*/ | 11346 PUT_NEWQUAD
|  NEWQUAD(NEWQUADNO + 1 ) = OPRND1;                                 /*NEWQUAD*/ | 11374 PUT_NEWQUAD
|  NEWQUAD(NEWQUADNO + 2 ) = OPRND2;                                 /*NEWQUAD*/ | 11394 PUT_NEWQUAD
|  NEWQUAD(NEWQUADNO + 3 ) = RSLT;                                   /*NEWQUAD*/ | 11414 PUT_NEWQUAD
|  NEWQUADNO = NEWQUADNO + 4;                                        /*NEWQUAD*/ | 11434 PUT_NEWQUAD
|  END;                                                              /*NEWQUAD*/ | 11446 PUT_NEWQUAD
|  SYMB_INDEX:PROCEDURE;                                             /*NEWQUAD*/ | 11452
|                                                                                | 11452 SYMB_INDEX
| /*SYMB_INDEX  RETURNS SYMBOL TABLE INDEX OF TEMP_CHAR   */                     | 11452 SYMB_INDEX
|                                                                                | 11452 SYMB_INDEX
|  DECLARE INDX FIXED;                                               /*NEWQUAD*/ | 11452 SYMB_INDEX
|  INDX = 1;                                                         /*NEWQUAD*/ | 11464 SYMB_INDEX
|  DO WHILE TEMP_CHAR ¬= SYMB(INDX);                                 /*NEWQUAD*/ | 11472 SYMB_INDEX
|     INDX = INDX + 1;                                               /*NEWQUAD*/ | 11522 SYMB_INDEX
|     END;                                                           /*NEWQUAD*/ | 11534 SYMB_INDEX
|  RETURN INDX;                                                      /*NEWQUAD*/ | 11542 SYMB_INDEX
|  END;                                                              /*NEWQUAD*/ | 11552 SYMB_INDEX
| LAB_INDEX:PROCEDURE;                                               /*NEWQUAD*/ | 11558
|                                                                                | 11558 LAB_INDEX
| /*LAB_INDEX  RETURN LABEL TABLE INDEX OF TEMP_CHAR   */                        | 11558 LAB_INDEX
|                                                                                | 11558 LAB_INDEX
|  DECLARE INDX FIXED;                                               /*NEWQUAD*/ | 11558 LAB_INDEX
|  INDX = 1;                                                         /*NEWQUAD*/ | 11570 LAB_INDEX
|  DO WHILE TEMP_CHAR ¬= LABID(INDX);                                /*NEWQUAD*/ | 11578 LAB_INDEX
|     INDX = INDX + 1;                                               /*NEWQUAD*/ | 11628 LAB_INDEX
|     END;                                                           /*NEWQUAD*/ | 11640 LAB_INDEX
|  RETURN INDX;                                                      /*NEWQUAD*/ | 11648 LAB_INDEX
|  END;                                                              /*NEWQUAD*/ | 11658 LAB_INDEX
| CONVAL_INDEX:PROCEDURE;                                            /*NEWQUAD*/ | 11664
|                                                                                | 11664 CONVAL_INDE
| /*CONVAL_INDEX  RETURNS CONSTANT TABLE INDEX OF TEMP_CHAR    */                | 11664 CONVAL_INDE
|                                                                                | 11664 CONVAL_INDE
|  DECLARE INDX FIXED;                                               /*NEWQUAD*/ | 11664 CONVAL_INDE
|  INDX=1;                                                           /*NEWQUAD*/ | 11676 CONVAL_INDE
|  DO WHILE TEMP_CHAR ¬= CONVAL(INDX);                               /*NEWQUAD*/ | 11684 CONVAL_INDE
|     INDX=INDX+1;                                                   /*NEWQUAD*/ | 11750 CONVAL_INDE
|     END;                                                           /*NEWQUAD*/ | 11762 CONVAL_INDE
|     RETURN INDX;                                                   /*NEWQUAD*/ | 11770 CONVAL_INDE
|  END;                                                              /*NEWQUAD*/ | 11780 CONVAL_INDE
| READQUAD:PROCEDURE(QUADNO);                                        /*NEWQUAD*/ | 11786
|                                                                                | 11786 READQUAD
| /*READQUAD PRINT OUT QUADS  */                                                 | 11786 READQUAD
|                                                                                | 11786 READQUAD
|  DECLARE QUADNO FIXED;                                             /*NEWQUAD*/ | 11786 READQUAD
|  DECLARE LATEST FIXED INITIAL(0);                                  /*NEWQUAD*/ | 11798 READQUAD
|  DECLARE(OPND1,OPND2,RES)FIXED;                                    /*NEWQUAD*/ | 11798 READQUAD
|  RCD_NR=(QUADNO-1)/MAXQUADS;                                       /*NEWQUAD*/ | 11798 READQUAD
|  IF RCD_NR ¬= RCD_BUFF THEN DO;                                    /*NEWQUAD*/ | 11820 READQUAD
|     QUADS=FILE(1,RCD_BUFF);                                        /*NEWQUAD*/ | 11836 READQUAD
|     RCD_BUFF=RCD_BUFF+1;                                           /*NEWQUAD*/ | 11866 READQUAD
|     END;                                                           /*NEWQUAD*/ | 11878 READQUAD
|  LOC_QUAD=((QUADNO-1) MOD MAXQUADS)*4 +1;                          /*NEWQUAD*/ | 11878 READQUAD
|  OPER = QUADS(LOC_QUAD);                                           /*NEWQUAD*/ | 11910 READQUAD
|  OPND1=QUADS(LOC_QUAD+1);                                          /*NEWQUAD*/ | 11926 READQUAD
|  OPND2=QUADS(LOC_QUAD+2);                                          /*NEWQUAD*/ | 11946 READQUAD
|  RES=QUADS(LOC_QUAD+3);                                            /*NEWQUAD*/ | 11966 READQUAD
|  IF QUADNO > LATEST THEN DO;                                       /*NEWQUAD*/ | 11986 READQUAD
|     LATEST = QUADNO;                                               /*NEWQUAD*/ | 12002 READQUAD
|     IF OPER = LAB THEN                                             /*NEWQUAD*/ | 12010 READQUAD
|        OPERAND1=LABID(OPND1);                                      /*NEWQUAD*/ | 12034 READQUAD
|     ELSE DO;                                                       /*NEWQUAD*/ | 12042 READQUAD
|        IF OPND1<0 THEN                                     /*NEWQUAD*/ |
|           OPERAND1=CONVAL(-OPND1);                          /*NEWQUAD*/ | 12074 READQUAD
|        ELSE                                                /*NEWQUAD*/ | 12092 READQUAD
|           IF OPND1 > 0 THEN                                /*NEWQUAD*/ | 12092 READQUAD
|              OPERAND1 = SYMB(OPND1);                       /*NEWQUAD*/ | 12124 READQUAD
|           ELSE                                             /*NEWQUAD*/ | 12132 READQUAD
|              OPERAND1='0';                                 /*NEWQUAD*/ | 12132 READQUAD
|        END;                                                /*NEWQUAD*/ | 12148 READQUAD
|     IF OPND2 < 0 THEN                                      /*NEWQUAD*/ | 12148 READQUAD
|        OPERAND2=CONVAL(-OPND2);                            /*NEWQUAD*/ | 12172 READQUAD
|     ELSE                                                   /*NEWQUAD*/ | 12190 READQUAD
|        IF OPND2 > 0 THEN                                   /*NEWQUAD*/ | 12190 READQUAD
|           OPERAND2 = SYMB(OPND2);                          /*NEWQUAD*/ | 12222 READQUAD
|        ELSE                                                /*NEWQUAD*/ | 12230 READQUAD
|           OPERAND2='0';                                    /*NEWQUAD*/ | 12230 READQUAD
|        IF OPER > HALT THEN DO;                             /*NEWQUAD*/ | 12246 READQUAD
|           IF OPER < REL THEN                               /*NEWQUAD*/ | 12262 READQUAD
|              RESULT = LABID(RES);                          /*NEWQUAD*/ | 12286 READQUAD
|           ELSE                                             /*NEWQUAD*/ | 12294 READQUAD
|              RESULT=SYMB(RES);                             /*NEWQUAD*/ | 12294 READQUAD
|     END;                                                   /*NEWQUAD*/ | 12318 READQUAD
|     IF OPER <= HALT THEN DO;                               /*NEWQUAD*/ | 12318 READQUAD
|        IF RES¬=0 THEN                                      /*NEWQUAD*/ | 12334 READQUAD
|           RESULT = SYMB(RES);                              /*NEWQUAD*/ | 12358 READQUAD
|           END;                                             /*NEWQUAD*/ | 12366 READQUAD
|     IF RES=0 THEN                                          /*NEWQUAD*/ | 12366 READQUAD
|        RESULT='0';                                         /*NEWQUAD*/ | 12390 READQUAD
|     BUFFER=PAD('    '||ALPHA(OPER),13);                    /*NEWQUAD*/ | 12390 READQUAD
|     BUFFER=PAD(BUFFER||OPERAND1,24);                       /*NEWQUAD*/ | 12438 READQUAD
|     BUFFER=PAD(BUFFER||OPERAND2,34);                       /*NEWQUAD*/ | 12478 READQUAD
|     BUFFER=PAD(BUFFER||RESULT,48);                         /*NEWQUAD*/ | 12518 READQUAD
|     BUFFER=PAD(BUFFER||RCD_NR,57);                         /*NEWQUAD*/ | 12558 READQUAD
|     BUFFER=PAD(BUFFER||LOC_QUAD,80);                       /*NEWQUAD*/ | 12606 READQUAD
|     OUTPUT=BUFFER;                                         /*NEWQUAD*/ | 12654 READQUAD
|     END;                                                   /*NEWQUAD*/ | 12674 READQUAD
|  END;                                                      /*NEWQUAD*/ | 12674 READQUAD
| PRINT_NEWQUADS:PROCEDURE;                                  /*NEWQUAD*/ | 12680
|                                                                        | 12680 PRINT_NEWQUAD
| /*PRINT_NEWQUADS  PRINT OUT REGISTER QUADS   */                        | 12680 PRINT_NEWQUAD
|                                                                        | 12680 PRINT_NEWQUAD
|  DECLARE CASE_NUM(30) FIXED INITIAL(0,                     /*NEWQUAD*/ | 12680 PRINT_NEWQUAD
|                                     1,                     /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
|                                     1,                     /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
|                                     1,                     /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
|                                     0,                     /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
|                                     0,                     /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
|                                     3,                     /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
|                                     6,                     /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
|                                     2,                     /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
|                                     2,                     /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
|                                     0,                     /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
|                                     1,                     /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
|                                     1,                     /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
|                                     1,                     /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
|                                     0,                     /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
|                                     0,                     /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
|                                     0,                     /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
|                                     0,                     /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
|                                     0,                     /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
|                                     0,                     /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
|                                     0,                     /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
|                                     0,                     /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
|                                     0,                     /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
|                                     0,                     /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
|                                     4,                     /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
|                                     7,                     /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
```

```
                                   8,                                    /*NEWQUAD*/ | 12692 PRINT_NEWQUA
                                   7,                                    /*NEWQUAD*/ | 12692 PRINT_NEWQUA
                                   0,                                    /*NEWQUAD*/ | 12692 PRINT_NEWQUA
                                   0,                                    /*NEWQUAD*/ | 12692 PRINT_NEWQUA
                                   0);                                   /*NEWQUAD*/ | 12692 PRINT_NEWQUA
OUTPUT(1) = '1                NEW QUADS';                                /*NEWQUAD*/ | 12692 PRINT_NEWQUA
DOUBLE_SPACE;                                                            /*NEWQUAD*/ | 12716 PRINT_NEWQUA
OUTPUT = 'OPERATOR  OPERAND1/  OPERAND2  RESULT';                        /*NEWQUAD*/ | 12740 PRINT_NEWQUA
OUTPUT = '          CONDITION            LABEL';                         /*NEWQUAD*/ | 12760 PRINT_NEWQUA
BR_FLAG=FALSE;                                                           /*NEWQUAD*/ | 12780 PRINT_NEWQUA
CURR_MIC=0;                                                              /*NEWQUAD*/ | 12786 PRINT_NEWQUA
BUFFER='                            GENERATED MICRO WORDS';              /*NEWQUAD*/ | 12792 PRINT_NEWQUA
BUFFER=PAD(BUFFER,80);                                                   /*NEWQUAD*/ | 12800 PRINT_NEWQUA
OUTPUT(3)=BUFFER;                                                        /*NEWQUAD*/ | 12824 PRINT_NEWQUA
BUFFER=PAD(X70,80);                                                      /*NEWQUAD*/ | 12848 PRINT_NEWQUA
OUTPUT(3)=BUFFER;                                                        /*NEWQUAD*/ | 12872 PRINT_NEWQUA
BUFFER='   L   C  C W  C C C   B   D    S    A    S'||                   /*NEWQUAD*/ | 12896 PRINT_NEWQUA
       '   S   S  S    U    S    R      U';                              /*NEWQUAD*/ | 12896 PRINT_NEWQUA
BUFFER=PAD(BUFFER,80);                                                   /*NEWQUAD*/ | 12920 PRINT_NEWQUA
OUTPUT(3)=BUFFER;                                                        /*NEWQUAD*/ | 12944 PRINT_NEWQUA
BUFFER='   O   L  I R  B D B   U   A    P    L    B'||                   /*NEWQUAD*/ | 12968 PRINT_NEWQUA
       '   B   D  B    B    R    I      P';                              /*NEWQUAD*/ | 12968 PRINT_NEWQUA
BUFFER=PAD(BUFFER,80);                                                   /*NEWQUAD*/ | 12992 PRINT_NEWQUA
OUTPUT(3)=BUFFER;                                                        /*NEWQUAD*/ | 13016 PRINT_NEWQUA
BUFFER='   C   K  R        A   S   D    S    U    C'||                   /*NEWQUAD*/ | 13040 PRINT_NEWQUA
       '   M   M  A    F    X    F      F';                              /*NEWQUAD*/ | 13040 PRINT_NEWQUA
BUFFER=PAD(BUFFER,80);                                                   /*NEWQUAD*/ | 13064 PRINT_NEWQUA
OUTPUT(3)=BUFFER;                                                        /*NEWQUAD*/ | 13088 PRINT_NEWQUA
DO II=0 TO NEWQUADNO-1 BY 4;                                             /*NEWQUAD*/ | 13112 PRINT_NEWQUA
   OPER=NEWQUAD(II);                                                     /*NEWQUAD*/ | 13158 PRINT_NEWQUA
   OPERATION=ALPHA(OPER);                                                /*NEWQUAD*/ | 13174 PRINT_NEWQUA
   DO CASE CASE_NUM(OPER);                                               /*NEWQUAD*/ | 13190 PRINT_NEWQUA
      DO;                                                                /*NEWQUAD*/ | 13222 PRINT_NEWQUA
         OPERAND1='R'||NEWQUAD(II+1);                                    /*NEWQUAD*/ | 13222 PRINT_NEWQUA
         OPERAND2='0';                                                   /*NEWQUAD*/ | 13266 PRINT_NEWQUA
         RESULT='R'||NEWQUAD(II+3);                                      /*NEWQUAD*/ | 13274 PRINT_NEWQUA
         END;                                                            /*NEWQUAD*/ | 13318 PRINT_NEWQUA
      DO;                                                                /*NEWQUAD*/ | 13318 PRINT_NEWQUA
         OPERAND1='R'||NEWQUAD(II+1);                                    /*NEWQUAD*/ | 13326 PRINT_NEWQUA
         OPERAND2='R'||NEWQUAD(II+2);                                    /*NEWQUAD*/ | 13370 PRINT_NEWQUA
         RESULT='R'||NEWQUAD(II+3);                                      /*NEWQUAD*/ | 13414 PRINT_NEWQUA
         END;                                                            /*NEWQUAD*/ | 13458 PRINT_NEWQUA
      DO;                                                                /*NEWQUAD*/ | 13458 PRINT_NEWQUA
         JJ=NEWQUAD(II+3);                                               /*NEWQUAD*/ | 13466 PRINT_NEWQUA
         OPERAND1='R'||NEWQUAD(II+1);                                    /*NEWQUAD*/ | 13486 PRINT_NEWQUA
         OPERAND2='0';                                                   /*NEWQUAD*/ | 13530 PRINT_NEWQUA
         RESULT=LABID(JJ);                                               /*NEWQUAD*/ | 13538 PRINT_NEWQUA
         END;                                                            /*NEWQUAD*/ | 13554 PRINT_NEWQUA
      DO;                                                                /*NEWQUAD*/ | 13554 PRINT_NEWQUA
         OPERAND1='0';                                                   /*NEWQUAD*/ | 13562 PRINT_NEWQUA
         OPERAND2='0';                                                   /*NEWQUAD*/ | 13570 PRINT_NEWQUA
         RESULT='0';                                                     /*NEWQUAD*/ | 13578 PRINT_NEWQUA
         END;                                                            /*NEWQUAD*/ | 13586 PRINT_NEWQUA
      DO;                                                                /*NEWQUAD*/ | 13586 PRINT_NEWQUA
         JJ=NEWQUAD(II+1);                                               /*NEWQUAD*/ | 13594 PRINT_NEWQUA
         OPERAND1=LABID(JJ);                                             /*NEWQUAD*/ | 13614 PRINT_NEWQUA
         OPERAND2='0';                                                   /*NEWQUAD*/ | 13630 PRINT_NEWQUA
         RESULT='0';                                                     /*NEWQUAD*/ | 13638 PRINT_NEWQUAD
         END;                                                            /*NEWQUAD*/ | 13646 PRINT_NEWQUAD
      ;                                                                  /*NEWQUAD*/ | 13646 PRINT_NEWQUAD
      DO;                                                                /*NEWQUAD*/ | 13654 PRINT_NEWQUAD
         JJ=NEWQUAD(II+3);                                               /*NEWQUAD*/ | 13662 PRINT_NEWQUAD
         OPERAND1='0';                                                   /*NEWQUAD*/ | 13682 PRINT_NEWQUAD
         OPERAND2='0';                                                   /*NEWQUAD*/ | 13690 PRINT_NEWQUAD
         RESULT=LABID(JJ);                                               /*NEWQUAD*/ | 13698 PRINT_NEWQUAD
         END;                                                            /*NEWQUAD*/ | 13714 PRINT_NEWQUAD
```

```
            DO;                                                   /*NEWQUAD*/  | 13714 PRINT_NEWQUAD
               JJ=NEWQUAD(II+1);                                  /*NEWQUAD*/  | 13722 PRINT_NEWQUAD
               IF JJ>0 THEN                                       /*NEWQUAD*/  | 13742 PRINT_NEWQUAD
                  OPERAND1=SYMB(JJ);                              /*NEWQUAD*/  | 13766 PRINT_NEWQUAD
               ELSE                                               /*NEWQUAD*/  | 13774 PRINT_NEWQUAD
                  OPERAND1=CONVAL(-JJ);                           /*NEWQUAD*/  | 13774 PRINT_NEWQUAD
               OPERAND2='0';                                      /*NEWQUAD*/  | 13808 PRINT_NEWQUAD
               RESULT='R'||NEWQUAD(II+3);                         /*NEWQUAD*/  | 13816 PRINT_NEWQUAD
               END;                                               /*NEWQUAD*/  | 13860 PRINT_NEWQUAD
            DO;                                                   /*NEWQUAD*/  | 13860 PRINT_NEWQUAD
               JJ=NEWQUAD(II+3);                                  /*NEWQUAD*/  | 13868 PRINT_NEWQUAD
               OPERAND1='R'||NEWQUAD(II+1);                       /*NEWQUAD*/  | 13886 PRINT_NEWQUAD
               OPERAND2='0';                                      /*NEWQUAD*/  | 13932 PRINT_NEWQUADS
               RESULT=SYMB(JJ);                                   /*NEWQUAD*/  | 13940 PRINT_NEWQUADS
               END;                                               /*NEWQUAD*/  | 13956 PRINT_NEWQUADS
            END;                                                  /*NEWQUAD*/  | 13956 PRINT_NEWQUADS
         BUFFER=PAD('    '||OPERATION,13);                        /*NEWQUAD*/  | 13964 PRINT_NEWQUADS
         BUFFER=PAD(BUFFER||OPERAND1,24);                         /*NEWQUAD*/  | 14004 PRINT_NEWQUADS
         BUFFER=PAD(BUFFER||OPERAND2,34);                         /*NEWQUAD*/  | 14044 PRINT_NEWQUADS
         BUFFER=PAD(BUFFER||RESULT,48);                           /*NEWQUAD*/  | 14084 PRINT_NEWQUADS
         OUTPUT = BUFFER;                                         /*NEWQUAD*/  | 14124 PRINT_NEWQUADS
         CALL MIC_GEN;                                            /*NEWQUAD*/  | 14144 PRINT_NEWQUADS
         END;                                                     /*NEWQUAD*/  | 14152 PRINT_NEWQUADS
   END;                                                           /*NEWQUAD*/  | 14160 PRINT_NEWQUADS
 SORT:PROCEDURE( A, B);                                           /*NEWQUAD*/  | 14166
                                                                               | 14166 SORT
 /*SORT   SORT SORTNUM OR SORTREF ARRAY    */                                  | 14166 SORT
                                                                               | 14166 SORT
   DECLARE (B, SWITCH) BIT(8);                                    /*NEWQUAD*/  | 14166 SORT
DECLARE(A,C,D,TEMP)FIXED;                                         /*NEWQUAD*/  | 14178 SORT
   SWITCH = TRUE;                                                 /*NEWQUAD*/  | 14178 SORT
   D = 0;                                                         /*NEWQUAD*/  | 14186 SORT
   DO WHILE SWITCH = TRUE;                                        /*NEWQUAD*/  | 14192 SORT
      SWITCH = FALSE;                                             /*NEWQUAD*/  | 14210 SORT
      DO C = D TO A - 1;                                          /*NEWQUAD*/  | 14216 SORT
         IF B= TRUE THEN DO;                                      /*NEWQUAD*/  | 14264 SORT
            IF SORTNUM(C) > SORTNUM( C + 1 ) THEN DO;             /*NEWQUAD*/  | 14282 SORT
               TEMP = SORTNUM(C);                                 /*NEWQUAD*/  | 14313 SORT
               SORTNUM(C) = SORTNUM(C+1);                         /*NEWQUAD*/  | 14334 SORT
               SORTNUM(C+1)=TEMP;                                 /*NEWQUAD*/  | 14362 SORT
               TEMP = SORTREF(C);                                 /*NEWQUAD*/  | 14382 SORT
               SORTREF(C) = SORTREF(C+1);                         /*NEWQUAD*/  | 14398 SORT
               SORTREF(C+1)= TEMP;                                /*NEWQUAD*/  | 14426 SORT
               TEMP=REGNO(C);                                     /*NEWQUAD*/  | 14446 SORT
               REGNO(C)= REGNO(C+1);                              /*NEWQUAD*/  | 14462 SORT
               REGNO(C+1)=TEMP;                                   /*NEWQUAD*/  | 14490 SORT
               SWITCH=TRUE;                                       /*NEWQUAD*/  | 14510 SORT
               END;                                               /*NEWQUAD*/  | 14518 SORT
            END;                                                  /*NEWQUAD*/  | 14518 SORT
         ELSE DO;                                                 /*NEWQUAD*/  | 14518 SORT
            IF SORTREF(C) > SORTREF(C+1) THEN DO;                 /*NEWQUAD*/  | 14526 SORT
               TEMP=SORTNUM(C);                                   /*NEWQUAD*/  | 14562 SORT
               SORTNUM(C)=SORTNUM(C+1);                           /*NEWQUAD*/  | 14578 SORT
               SORTNUM(C+1)=TEMP;                                 /*NEWQUAD*/  | 14606 SORT
               TEMP=SORTREF(C);                                   /*NEWQUAD*/  | 14626 SORT
               SORTREF(C)=SORTREF(C+1);                           /*NEWQUAD*/  | 14642 SORT
               SORTREF(C+1)=TEMP;                                 /*NEWQUAD*/  | 14670 SORT
               TEMP=REGNO(C);                                     /*NEWQUAD*/  | 14690 SORT
               REGNO(C)=REGNO(C+1);                               /*NEWQUAD*/  | 14706 SORT
               REGNO(C+1)=TEMP;                                   /*NEWQUAD*/  | 14734 SORT
               SWITCH=TRUE;                                       /*NEWQUAD*/  | 14754 SORT
               END;                                               /*NEWQUAD*/  | 14762 SORT
            END;                                                  /*NEWQUAD*/  | 14762 SORT
         END;                                                     /*NEWQUAD*/  | 14762 SORT
      D=D+1;                                                      /*NEWQUAD*/  | 14770 SORT
      END;                                                        /*NEWQUAD*/  | 14782 SORT
      D = 0;                                                          /*NEWQUAD*/ | 14790 SORT
      DO C = 0 TO A - 1;                                              /*NEWQUAD*/ | 14796 SORT
         IF B=TRUE THEN DO;                                           /*NEWQUAD*/ | 14842 SORT
            IF SORTNUM(C) = SORTNUM(C+1) THEN                         /*NEWQUAD*/ | 14860 SORT
               D = D + 1;                                             /*NEWQUAD*/ | 14904 SORT
            ELSE                                                      /*NEWQUAD*/ | 14908 SORT
               RETURN D;                                              /*NEWQUAD*/ | 14908 SORT
            END;                                                      /*NEWQUAD*/ | 14926 SORT
         ELSE DO;                                                     /*NEWQUAD*/ | 14926 SORT
            IF SORTREF(C) = SORTREF(C+1) THEN                         /*NEWQUAD*/ | 14934 SORT
               D = D + 1;                                             /*NEWQUAD*/ | 14978 SORT
            ELSE                                                      /*NEWQUAD*/ | 14982 SORT
               RETURN D;                                              /*NEWQUAD*/ | 14982 SORT
            END;                                                      /*NEWQUAD*/ | 15000 SORT
         END;                                                         /*NEWQUAD*/ | 15000 SORT
      RETURN D;                                                       /*NEWQUAD*/ | 15009 SORT
      END;                                                            /*NEWQUAD*/ | 15018 SORT
DEALLOCATE:PROCEDURE;                                                 /*NEWQUAD*/ | 15024
                                                                                  | 15024 DEALLOCAT
/*DEALLOCTE  DEALLOCTES A REGISTER FROM VARIABLES ASSIGNED TO IT                  | 15024 DEALLOCAT
                1- DEALLOCATE THE REGISTER WITH NO TEMPORARY VARIABLE AND         | 15024 DEALLOCAT
                   LEAST NUMBER OF VARIABLE.                                      | 15024 DEALLOCAT
               2- IF TWO REGISTER HAVE THE SAME NUMBER OF VARIABLES ASSIGNED      | 15024 DEALLOCAT
                   TO THEM, THEN PICK THE ONE THAT HAS NOT BEEN REFRENCED         | 15024 DEALLOCAT
                   LATELY.                                                        | 15024 DEALLOCAT
                3- IF NO REGISTER IS FOUND WITH NO TEMPORARY VARIABLE, THEN       | 15024 DEALLOCAT
                   PICK THE REGISTER THAT HAS LEAST NUMBER OF TEMPORARY VARIABLE  | 15024 DEALLOCAT
                         */                                                       | 15024 DEALLOCAT
                                                                                  | 15024 DEALLOCAT
 DECLARE(TEMP,REG,MINIMUM)FIXED;                                      /*NEWQUAD*/ | 15024 DEALLOCAT
 DECLARE TEMPVARNUM(6) FIXED;                                         /*NEWQUAD*/ | 15036 DEALLOCAT
 DECLARE TEMPREGNO(6) FIXED;                                          /*NEWQUAD*/ | 15036 DEALLOCAT
 KK = 0;                                                              /*NEWQUAD*/ | 15036 DEALLOCAT
 JJ = 0;                                                              /*NEWQUAD*/ | 15042 DEALLOCAT
 DO II = 1 TO MAXREG;                                                 /*NEWQUAD*/ | 15048 DEALLOCAT
    IF DEALLOCABL(II) = FALSE THEN DO;                                /*NEWQUAD*/ | 15092 DEALLOCAT
       IF TEMP(II) = 0 THEN DO;                                       /*NEWQUAD*/ | 15114 DEALLOCAT
          IF VARNUM(II) = 1 THEN DO;                                  /*NEWQUAD*/ | 15138 DEALLOCAT
             LL = POINT(II);                                          /*NEWQUAD*/ | 15162 DEALLOCAT
             IF BYTE(VARIBLS(LL),0) >= BYTE('0') &                    /*NEWQUAD*/ | 15178 DEALLOCAT
                BYTE(VARIBLS(LL),0) <= BYTE('9') THEN DO;             /*NEWQUAD*/ | 15216 DEALLOCAT
                NEXT_VAR(LL) = LAVS;                                  /*NEWQUAD*/ | 15268 DEALLOCAT
                LAVS = LL;                                            /*NEWQUAD*/ | 15284 DEALLOCAT
                RD = TRUE;                                            /*NEWQUAD*/ | 15292 DEALLOCAT
                RETURN II;                                            /*NEWQUAD*/ | 15300 DEALLOCAT
                END;                                                  /*NEWQUAD*/ | 15310 DEALLOCAT
             ELSE DO;                                                 /*NEWQUAD*/ | 15310 DEALLOCAT
                IF CHANGE(LL) = FALSE THEN DO;                        /*NEWQUAD*/ | 15318 DEALLOCAT
                   NEXT_VAR(LL) = LAVS;                               /*NEWQUAD*/ | 15340 DEALLOCAT
                  LAVS = LL;                                          /*NEWQUAD*/ | 15356 DEALLOCAT
                   RD = TRUE;                                         /*NEWQUAD*/ | 15364 DEALLOCAT
                   RETURN II;                                         /*NEWQUAD*/ | 15372 DEALLOCAT
                   END;                                               /*NEWQUAD*/ | 15382 DEALLOCAT
                SORTREF(JJ) = REFRENCE(II);                           /*NEWQUAD*/ | 15382 DEALLOCAT
                SORTNUM(JJ) = VARNUM(II);                             /*NEWQUAD*/ | 15406 DEALLOCAT
                REGNO(JJ) = II;                                       /*NEWQUAD*/ | 15430 DEALLOCAT
                JJ = JJ + 1;                                          /*NEWQUAD*/ | 15446 DEALLOCAT
                END;                                                  /*NEWQUAD*/ | 15458 DEALLOCAT
             END;                                                     /*NEWQUAD*/ | 15458 DEALLOCAT
          ELSE DO;                                                    /*NEWQUAD*/ | 15458 DEALLOCAT
             SORTREF(JJ) = REFRENCE(II);                              /*NEWQUAD*/ | 15466 DEALLOCAT
             SORTNUM(JJ) = VARNUM(II);                                /*NEWQUAD*/ | 15490 DEALLOCAT
             REGNO(JJ) = II;                                          /*NEWQUAD*/ | 15514 DEALLOCAT
             JJ = JJ +1;                                              /*NEWQUAD*/ | 15530 DEALLOCAT
             END;                                                     /*NEWQUAD*/ | 15542 DEALLOCAT
          END;                                                        /*NEWQUAD*/ | 15542 DEALLOCAT
          ELSE DO;                                                   /*NEWQUAD*/  | 15542 DEALLOCATE
             TEMPVARNUM(KK) = VARNUM(II);                            /*NEWQUAD*/  | 15550 DEALLOCATE
             TEMPREGNO(KK) = II;                                     /*NEWQUAD*/  | 15574 DEALLOCATE
             KK = KK + 1;                                            /*NEWQUAD*/  | 15590 DEALLOCATE
             END;                                                    /*NEWQUAD*/  | 15602 DEALLOCATE
          END;                                                       /*NEWQUAD*/  | 15602 DEALLOCATE
       END;                                                          /*NEWQUAD*/  | 15602 DEALLOCATE
    IF JJ = 0 THEN DO;                                               /*NEWQUAD*/  | 15610 DEALLOCATE
      MINIMUM = 100;                                                 /*NEWQUAD*/  | 15626 DEALLOCATE
      DO II = 0 TO KK -1;                                            /*NEWQUAD*/  | 15634 DEALLOCATE
          IF TEMPVARNUM(II) < MINIMUM THEN DO;                       /*NEWQUAD*/  | 15680 DEALLOCATE
             MINIMUM = TEMPVARNUM(II);                               /*NEWQUAD*/  | 15704 DEALLOCATE
             REG = TEMPREGNO(II);                                    /*NEWQUAD*/  | 15720 DEALLOCATE
             END;                                                    /*NEWQUAD*/  | 15736 DEALLOCATE
          END;                                                       /*NEWQUAD*/  | 15736 DEALLOCATE
       END;                                                          /*NEWQUAD*/  | 15744 DEALLOCATE
    ELSE DO;                                                         /*NEWQUAD*/  | 15744 DEALLOCATE
       IF JJ ¬= 1 THEN DO;                                           /*NEWQUAD*/  | 15752 DEALLOCATE
          II = SORT(JJ-1,TRUE);                                      /*NEWQUAD*/  | 15768 DEALLOCATE
          KK= SORT(II,FALSE);                                        /*NEWQUAD*/  | 15800 DEALLOCATE
          REG = REGNO(0);                                            /*NEWQUAD*/  | 15826 DEALLOCATE
          END;                                                       /*NEWQUAD*/  | 15840 DEALLOCATE
       ELSE                                                          /*NEWQUAD*/  | 15840 DEALLOCATE
       REG = REGNO(0);                                               /*NEWQUAD*/  | 15840 DEALLOCATE
       END;                                                          /*NEWQUAD*/  | 15862 DEALLOCATE
    LL = POINT(REG);                                                 /*NEWQUAD*/  | 15862 DEALLOCATE
    DO II =  1 TO VARNUM(REG);                                       /*NEWQUAD*/  | 15878 DEALLOCATE
       IF CHANGE(LL) = TRUE THEN DO;                                 /*NEWQUAD*/  | 15930 DEALLOCATE
                                                                                  | 15952 DEALLOCATE
  /*GENERATE A WT REGISTER QUAD  */                                               | 15952 DEALLOCATE
                                                                                  | 15952 DEALLOCATE
          TEMP_CHAR=VARIBLS(LL);                                     /*NEWQUAD*/  | 15952 DEALLOCATE
          OPRTOR=26;                                                 /*NEWQUAD*/  | 15968 DEALLOCATE
          OPRND1=REG;                                                /*NEWQUAD*/  | 15976 DEALLOCATE
          OPRND2=0;                                                  /*NEWQUAD*/  | 15984 DEALLOCATE
          RSLT=SYMB_INDEX;                                           /*NEWQUAD*/  | 15990 DEALLOCATE
          CURR_MIC=CURR_MIC+4;                                       /*NEWQUAD*/  | 16002 DEALLOCATE
          CALL PUT_NEWQUAD;                                          /*NEWQUAD*/  | 16014 DEALLOCATE
          END;                                                       /*NEWQUAD*/  | 16022 DEALLOCATE
       TEMP = NEXT_VAR(LL);                                          /*NEWQUAD*/  | 16022 DEALLOCATE
       NEXT_VAR(LL) = LAVS;                                          /*NEWQUAD*/  | 16038 DEALLOCATE
       LAVS = LL;                                                    /*NEWQUAD*/  | 16054 DEALLOCATE
       LL = TEMP;                                                    /*NEWQUAD*/  | 16062 DEALLOCATE
       END;                                                          /*NEWQUAD*/  | 16070 DEALLOCATE
    RD = TRUE;                                                       /*NEWQUAD*/  | 16078 DEALLOCATE
    RETURN REG;                                                      /*NEWQUAD*/  | 16086 DEALLOCATE
    END;                                                             /*NEWQUAD*/  | 16096 DEALLOCATE
  DEALLOC_TEMP:PROCEDURE(OPERAND,REG);                               /*NEWQUAD*/  | 16102
                                                                                  | 16102 DEALLOC_TE
  /*DEALLOC_TEMP    DELETE TEMPORARY VARIABLE FROM LIST OF VARIABLES              | 16102 DEALLOC_TE
                    ASSIGNED TO THE REGISTER   */                                 | 16102 DEALLOC_TE
                                                                                  | 16102 DEALLOC_TE
    DECLARE(REG,PRE_VAR)FIXED;                                       /*NEWQUAD*/  | 16102 DEALLOC_TE
    DECLARE(OPERAND)CHARACTER;                                       /*NEWQUAD*/  | 16114 DEALLOC_TE
    DEALLOCABL(REG) = FALSE;                                         /*NEWQUAD*/  | 16114 DEALLOC_TE
    IF BYTE(OPERAND,0) = BYTE('.') THEN DO;                          /*NEWQUAD*/  | 16124 DEALLOC_TE
       LL = POINT(REG);                                              /*NEWQUAD*/  | 16146 DEALLOC_TE
       DO II = 1 TO VARNUM(REG);                                     /*NEWQUAD*/  | 16162 DEALLOC_TE
          IF VARIBLS(LL) = OPERAND THEN DO;                          /*NEWQUAD*/  | 16214 DEALLOC_TE
             IF LL =POINT(REG) THEN                                  /*NEWQUAD*/  | 16264 DEALLOC_TE
                POINT(REG) = NEXT_VAR(LL);                           /*NEWQUAD*/  | 16296 DEALLOC_TE
             ELSE                                                    /*NEWQUAD*/  | 16312 DEALLOC_TE
                NEXT_VAR(PRE_VAR) = NEXT_VAR(LL);                    /*NEWQUAD*/  | 16312 DEALLOC_TE
             NEXT_VAR(LL) = LAVS;                                    /*NEWQUAD*/  | 16344 DEALLOC_TE
             LAVS = LL;                                              /*NEWQUAD*/  | 16360 DEALLOC_TE
             II = VARNUM(REG) + 1;                                   /*NEWQUAD*/  | 16368 DEALLOC_TE
```

```
|         [illegible]                                                  /*NEWQUAD*/ | [illegible] DEALLOC_
|         ELSE DO:                                                     /*NEWQUAD*/ | 16388 DEALLOC_TEM
|            PRE_VAR = LL;                                             /*NEWQUAD*/ | 16396 DEALLOC_TEM
|            LL = NEXT_VAR(LL);                                        /*NEWQUAD*/ | 16404 DEALLOC_TEM
|            END;                                                      /*NEWQUAD*/ | 16420 DEALLOC_TEM
|         END;                                                         /*NEWQUAD*/ | 16420 DEALLOC_TEM
|      IF VARNUM(REG) ¬= 1 THEN DO;                                    /*NEWQUAD*/ | 16428 DEALLOC_TEM
|         TEMP(REG) = TEMP(REG) - 1;                                   /*NEWQUAD*/ | 16452 DEALLOC_TEM
|         VARNUM(REG) = VARNUM(REG) - 1;                               /*NEWQUAD*/ | 16480 DEALLOC_TEM
|         END;                                                         /*NEWQUAD*/ | 16508 DEALLOC_TEM
|      ELSE                                                            /*NEWQUAD*/ | 16508 DEALLOC_TEM
|         STATUS(REG) = FALSE;                                         /*NEWQUAD*/ | 16508 DEALLOC_TEM
|      END;                                                            /*NEWQUAD*/ | 16526 DEALLOC_TEM
|   END;                                                               /*NEWQUAD*/ | 16526 DEALLOC_TEM
| FIND_RES:PROCEDURE(RESULT);                                          /*NEWQUAD*/ | 16532
|                                                                                  | 16532 FIND_RES
|  /*FIND_RES  SEARCH REGISTER TABLE FOR RESULT VARIABLE                            | 16532 FIND_RES
|              1- IF FOUND AND THE ONLY VARIABLE ASSIGNED TO THE REGISTER,         | 16532 FIND_RES
|                 THEN RETURN REG#                                                 | 16532 FIND_RES
|              2- IF FOUND AND MORE THAN ONE VARIABLE, THEN DELETE  VARIABLE       | 16532 FIND_RES
|                 FROM LIST OF VARIABLES ASSIGNED TO THE REGISTER AND GO TO 3.     | 16532 FIND_RES
|              3- RETURN REG# OF A FREE REGISTER , IF NOT FOUND RETURN -1  */      | 16532 FIND_RES
|                                                                                  | 16532 FIND_RES
|  DECLARE(RESULT)CHARACTER;                                           /*NEWQUAD*/ | 16532 FIND_RES
|  DECLARE(PRE_VAR)FIXED;                                              /*NEWQUAD*/ | 16544 FIND_RES
|  DO II = 1 TO MAXREG;                                                /*NEWQUAD*/ | 16544 FIND_RES
|     IF STATUS(II) = TRUE THEN DO;                                    /*NEWQUAD*/ | 16588 FIND_RES
|        LL = POINT(II);                                               /*NEWQUAD*/ | 16610 FIND_RES
|        DO JJ = 1 TO VARNUM(II);                                      /*NEWQUAD*/ | 16626 FIND_RES
|           IF VARIBLS(LL) = RESULT THEN DO:                           /*NEWQUAD*/ | 16678 FIND_RES
|              IF VARNUM(II) ¬= 1 THEN DO;                             /*NEWQUAD*/ | 16728 FIND_RES
|                 VARNUM(II) = VARNUM(II) -1;                          /*NEWQUAD*/ | 16752 FIND_RES
|                 IF BYTE(RESULT,0)=BYTE('-') THEN                     /*NEWQUAD*/ | 16780 FIND_RES
|                    TEMP(II) = TEMP(II) - 1;                          /*NEWQUAD*/ | 16810 FIND_RES
|                 IF LL = POINT(II) THEN                               /*NEWQUAD*/ | 16830 FIND_RES
|                    POINT(II) = NEXT_VAR(LL);                         /*NEWQUAD*/ | 16862 FIND_RES
|                 ELSE                                                 /*NEWQUAD*/ | 16878 FIND_RES
|                    NEXT_VAR(PRE_VAR) = NEXT_VAR(LL);                 /*NEWQUAD*/ | 16878 FIND_RES
|                 NEXT_VAR(LL) = LAVS;                                 /*NEWQUAD*/ | 16910 FIND_RES
|                 LAVS = LL;                                           /*NEWQUAD*/ | 16926 FIND_RES
|                 JJ = VARNUM(II) + 1;                                 /*NEWQUAD*/ | 16934 FIND_RES
|                 II = MAXREG + 1;                                     /*NEWQUAD*/ | 16954 FIND_RES
|                 END;                                                 /*NEWQUAD*/ | 16966 FIND_RES
|              ELSE DO;                                                /*NEWQUAD*/ | 16966 FIND_RES
|                 RD = FALSE;                                          /*NEWQUAD*/ | 16974 FIND_RES
|                 RETURN II;                                           /*NEWQUAD*/ | 16980 FIND_RES
|                 END;                                                 /*NEWQUAD*/ | 16990 FIND_RES
|              END;                                                    /*NEWQUAD*/ | 16990 FIND_RES
|           ELSE DO;                                                   /*NEWQUAD*/ | 16990 FIND_RES
|              PRE_VAR = LL;                                           /*NEWQUAD*/ | 16998 FIND_RES
|              LL = NEXT_VAR(LL);                                      /*NEWQUAD*/ | 17006 FIND_RES
|              END;                                                    /*NEWQUAD*/ | 17022 FIND_RES
|           END;                                                       /*NEWQUAD*/ | 17022 FIND_RES
|        END;                                                          /*NEWQUAD*/ | 17030 FIND_RES
|     END;                                                             /*NEWQUAD*/ | 17030 FIND_RES
|  RD = TRUE;                                                          /*NEWQUAD*/ | 17038 FIND_RES
|  DO II = 1 TO MAXREG;                                                /*NEWQUAD*/ | 17046 FIND_RES
|     IF STATUS(II) = FALSE THEN DO;                                   /*NEWQUAD*/ | 17090 FIND_RES
|        RETURN II;                                                    /*NEWQUAD*/ | 17112 FIND_RES
|        END;                                                          /*NEWQUAD*/ | 17122 FIND_RES
|     END;                                                             /*NEWQUAD*/ | 17122 FIND_RES
|  RETURN -1;                                                          /*NEWQUAD*/ | 17130 FIND_RES
|  END;                                                                /*NEWQUAD*/ | 17140 FIND_RES
| ALLOC_REG_RES:PROCEDURE(RESULT);                                     /*NEWQUAD*/ | 17146
|                                                                                  | 17146 ALLOC_REG_RE
|  /*ALLOC_REG_RES  ALLOCATE A REGISTER TO A RESULT VARIABLE      */               | 17146 ALLOC_REG_RE
```

```
                                                                              | 17146 ALLOC_REG_
   DECLARE(RESULT)CHARACTER;                                     /*NEWQUAD*/  | 17146 ALLOC_REG_RE
   DECLARE(REG)FIXED;                                            /*NEWQUAD*/  | 17158 ALLOC_REG_RE
   REG = FIND_RES(RESULT);                                       /*NEWQUAD*/  | 17158 ALLOC_REG_RE
   IF REG = -1 THEN                                              /*NEWQUAD*/  | 17178 ALLOC_REG_RE
      REG = DEALLOCATE;                                          /*NEWQUAD*/  | 17202 ALLOC_REG_RE
   IF RD = TRUE THEN DO;                                         /*NEWQUAD*/  | 17206 ALLOC_REG_RE
      STATUS(REG) = TRUE;                                        /*NEWQUAD*/  | 17224 ALLOC_REG_RE
      VARNUM(REG) = 1;                                           /*NEWQUAD*/  | 17236 ALLOC_REG_RE
      REFRENCE(REG) = REFNO;                                     /*NEWQUAD*/  | 17252 ALLOC_REG_RE
      REFNO = REFNO + 1;                                         /*NEWQUAD*/  | 17268 ALLOC_REG_RE
      DEALLOCABL(REG) = FALSE;                                   /*NEWQUAD*/  | 17280 ALLOC_REG_RE
      IF BYTE(RESULT,0)=BYTE('.') THEN                           /*NEWQUAD*/  | 17290 ALLOC_REG_RE
         TEMP(REG) = 1;                                          /*NEWQUAD*/  | 17320 ALLOC_REG_RE
      ELSE                                                       /*NEWQUAD*/  | 17328 ALLOC_REG_RE
         TEMP(REG) =0;                                           /*NEWQUAD*/  | 17328 ALLOC_REG_RE
      LL,POINT(REG) = LAVS;                                      /*NEWQUAD*/  | 17350 ALLOC_REG_RE
      LAVS = NEXT_VAR(LAVS);                                     /*NEWQUAD*/  | 17370 ALLOC_REG_RE
      VARIBLS(LL) = RESULT;                                      /*NEWQUAD*/  | 17386 ALLOC_REG_RE
      NEXT_VAR(LL) = 0;                                          /*NEWQUAD*/  | 17402 ALLOC_REG_RE
      CHANGE(LL) = TRUE;                                         /*NEWQUAD*/  | 17416 ALLOC_REG_RE
      END;                                                       /*NEWQUAD*/  | 17428 ALLOC_REG_RE
   ELSE DO;                                                      /*NEWQUAD*/  | 17428 ALLOC_REG_RE
      LL = POINT(REG);                                           /*NEWQUAD*/  | 17436 ALLOC_REG_RE
      CHANGE(LL) = TRUE;                                         /*NEWQUAD*/  | 17452 ALLOC_REG_RE
      END;                                                       /*NEWQUAD*/  | 17464 ALLOC_REG_RE
   RETURN REG;                                                   /*NEWQUAD*/  | 17464 ALLOC_REG_RE
   END;                                                          /*NEWQUAD*/  | 17474 ALLOC_REG_RE
 FIND_OP:PROCEDURE(OPERAND);                                     /*NEWQUAD*/  | 17480
                                                                              | 17480 FIND_OP
 /*FIND_OP  SEARCH REGISTER TABLE FOR OPERAND VARIABLE                        | 17480 FIND_OP
            1- IF FOUND RETURN REG#                                           | 17480 FIND_OP
            2- IF NOT FOUND RETURN REG# OF A FREE REGISTER                    | 17480 FIND_OP
            3- IF NO FREE REGISTER IS AVAILABLE RETURN -1         */          | 17480 FIND_OP
                                                                              | 17480 FIND_OP
   DECLARE(NOT_ALLOCATED)FIXED;                                  /*NEWQUAD*/  | 17480 FIND_OP
   DECLARE(OPERAND)CHARACTER;                                    /*NEWQUAD*/  | 17492 FIND_OP
   NOT_ALLOCATED = -1;                                           /*NEWQUAD*/  | 17492 FIND_OP
   DO II = 1 TO MAXREG;                                          /*NEWQUAD*/  | 17500 FIND_OP
      IF STATUS(II) = TRUE THEN DO;                              /*NEWQUAD*/  | 17544 FIND_OP
         LL = POINT(II);                                         /*NEWQUAD*/  | 17566 FIND_OP
         DO JJ = 1 TO VARNUM(II);                                /*NEWQUAD*/  | 17582 FIND_OP
            IF VARIBLS(LL) = OPERAND THEN DO;                    /*NEWQUAD*/  | 17634 FIND_OP
               RD = FALSE;                                       /*NEWQUAD*/  | 17684 FIND_OP
               RETURN II;                                        /*NEWQUAD*/  | 17690 FIND_OP
               END;                                              /*NEWQUAD*/  | 17700 FIND_OP
            ELSE                                                 /*NEWQUAD*/  | 17700 FIND_OP
               LL = NEXT_VAR(LL);                                /*NEWQUAD*/  | 17700 FIND_OP
            END;                                                 /*NEWQUAD*/  | 17724 FIND_OP
         END;                                                    /*NEWQUAD*/  | 17732 FIND_OP
      ELSE                                                       /*NEWQUAD*/  | 17732 FIND_OP
         NOT_ALLOCATED = II;                                     /*NEWQUAD*/  | 17732 FIND_OP
      END;                                                       /*NEWQUAD*/  | 17748 FIND_OP
   RD = TRUE;                                                    /*NEWQUAD*/  | 17756 FIND_OP
   RETURN NOT_ALLOCATED;                                         /*NEWQUAD*/  | 17764 FIND_OP
   END;                                                          /*NEWQUAD*/  | 17774 FIND_OP
 ALLOC_REG_OP:PROCEDURE(OPERAND);                                /*NEWQUAD*/  | 17780
                                                                              | 17780 ALLOC_REG_O
 /*ALLOC_REG_OP  ALLOCATE A REGISTER TO AND OPERAND VARIABLE                  | 17780 ALLOC_REG_O
                 1- IF ALREADY ASSIGNED RETURN REG#                           | 17780 ALLOC_REG_O
                 2- IF NOT GENERATE A RD OR RDAD REGISTER QUAD                | 17780 ALLOC_REG_O
                    AND RETURN REG#       */                                  | 17780 ALLOC_REG_O
                                                                              | 17780 ALLOC_REG_O
   DECLARE(REG)FIXED;                                            /*NEWQUAD*/  | 17780 ALLOC_REG_O
   DECLARE(OPERAND)CHARACTER;                                    /*NEWQUAD*/  | 17792 ALLOC_REG_O
   REG = FIND_OP(OPERAND);                                       /*NEWQUAD*/  | 17792 ALLOC_REG_O
```

```
  IF REG = -1 THEN                                            /*NEWQUAD*/ | 17812 ALLOC_REG_0
     REG = DEALLOCATE;                                        /*NEWQUAD*/ | 17836 ALLOC_REG_0
  DEALLOCABL(REG) = TRUE;                                     /*NEWQUAD*/ | 17840 ALLOC_REG_0
  REFERENCE(REG) = REFNO;                                     /*NEWQUAD*/ | 17852 ALLOC_REG_0
  REFNO = REFNO + 1;                                          /*NEWQUAD*/ | 17868 ALLOC_REG_0
  IF RD = TRUE THEN DO;                                       /*NEWQUAD*/ | 17880 ALLOC_REG_0
     STATUS(REG) = TRUE;                                      /*NEWQUAD*/ | 17898 ALLOC_REG_0
     VARNUM(REG) = 1;                                         /*NEWQUAD*/ | 17910 ALLOC_REG_0
     LL,POINT(REG) = LAVS;                                    /*NEWQUAD*/ | 17926 ALLOC_REG_0
     LAVS = NEXT_VAR(LAVS);                                   /*NEWQUAD*/ | 17946 ALLOC_REG_0
     VARIBLS(LL) = OPERAND;                                   /*NEWQUAD*/ | 17962 ALLOC_REG_0
     NEXT_VAR(LL) = 0;                                        /*NEWQUAD*/ | 17978 ALLOC_REG_0
     CHANGE(LL) = FALSE;                                      /*NEWQUAD*/ | 17992 ALLOC_REG_0
     IF BYTE(OPERAND,0) = BYTE('.') THEN                      /*NEWQUAD*/ | 18002 ALLOC_REG_0
        TEMP(REG) = 1;                                        /*NEWQUAD*/ | 18032 ALLOC_REG_0
     ELSE                                                     /*NEWQUAD*/ | 18040 ALLOC_REG_0
        TEMP(REG)=0;                                          /*NEWQUAD*/ | 18040 ALLOC_REG_0
     IF OPER=SUBS | OPER=SUBL THEN DO;                        /*NEWQUAD*/ | 18062 ALLOC_REG_0
        OPRTOR=27;                                            /*NEWQUAD*/ | 18124 ALLOC_REG_0
        TEMP_CHAR=OPERAND;                                    /*NEWQUAD*/ | 18132 ALLOC_REG_0
        OPRND1=SYMB_INDEX;                                    /*NEWQUAD*/ | 18140 ALLOC_REG_0
        OPRND2=0;                                             /*NEWQUAD*/ | 18152 ALLOC_REG_0
        RSLT=REG;                                             /*NEWQUAD*/ | 18158 ALLOC_REG_0
        CURR_MIC=CURR_MIC+2;                                  /*NEWQUAD*/ | 18166 ALLOC_REG_0
        CALL PUT_NEWQUAD;                                     /*NEWQUAD*/ | 18178 ALLOC_REG_0
        END;                                                  /*NEWQUAD*/ | 18186 ALLOC_REG_0
     ELSE  DO;                                                /*NEWQUAD*/ | 18186 ALLOC_REG_0
        TEMP_CHAR=OPERAND;                                    /*NEWQUAD*/ | 18194 ALLOC_REG_0
        IF BYTE(OPERAND,0)>= BYTE('0') &                      /*NEWQUAD*/ | 18202 ALLOC_REG_0
           BYTE(OPERAND,0) <= BYTE('9') THEN                  /*NEWQUAD*/ | 18232 ALLOC_REG_0
           OPRND1=-CONVAL_INDEX;                              /*NEWQUAD*/ | 18264 ALLOC_REG_0
        ELSE                                                  /*NEWQUAD*/ | 18290 ALLOC_REG_0
          OPRND1=SYMB_INDEX;                                  /*NEWQUAD*/ | 18290 ALLOC_REG_0
        OPRTOR=25;                                            /*NEWQUAD*/ | 18310 ALLOC_REG_0
        OPRND2=0;                                             /*NEWQUAD*/ | 18318 ALLOC_REG_0
        RSLT=REG;                                             /*NEWQUAD*/ | 18324 ALLOC_REG_0
        CURR_MIC=CURR_MIC+4;                                  /*NEWQUAD*/ | 18332 ALLOC_REG_0
        CALL PUT_NEWQUAD;                                     /*NEWQUAD*/ | 18344 ALLOC_REG_0
        END;                                                  /*NEWQUAD*/ | 18352 ALLOC_REG_0
     END;                                                     /*NEWQUAD*/ | 18352 ALLOC_REG_0
  RETURN REG;                                                 /*NEWQUAD*/ | 18352 ALLOC_REG_0
  END;      ;                                                 /*NEWQUAD*/ | 18362 ALLOC_REG_0
WRITE_REGS:PROCEDURE;                                         /*NEWQUAD*/ | 18368
                                                                          | 18368 WRITE_REGS
/*WRITE REGS  GENERATE WT REGISTER QUADS FOR VARIABLES ASSIGNED           | 18368 WRITE_REGS
              TO REGISTERS                   */                           | 18368 WRITE_REGS
                                                                          | 18368 WRITE_REGS
  DO II= 1 TO MAXREG;                                         /*NEWQUAD*/ | 18368 WRITE_REGS
     IF STATUS(II) = TRUE THEN DO;                            /*NEWQUAD*/ | 18424 WRITE_REGS
        LL = POINT(II);                                       /*NEWQUAD*/ | 18446 WRITE_REGS
        DO JJ = 1 TO VARNUM(II);                              /*NEWQUAD*/ | 18462 WRITE_REGS
           IF CHANGE(LL) = TRUE THEN DO;                      /*NEWQUAD*/ | 18514 WRITE_REGS
              CHANGE(LL) = FALSE;                             /*NEWQUAD*/ | 18536 WRITE_REGS
              TEMP_CHAR=VARIBLS(LL);                          /*NEWQUAD*/ | 18546 WRITE_REGS
              RSLT=SYMB_INDEX;                                /*NEWQUAD*/ | 18562 WRITE_REGS
              OPRTOR=26;                                      /*NEWQUAD*/ | 18574 WRITE_REGS
              OPRND1=II;                                      /*NEWQUAD*/ | 18582 WRITE_REGS
              OPRND2=0;                                       /*NEWQUAD*/ | 18590 WRITE_REGS
              CURR_MIC=CURR_MIC+4;                            /*NEWQUAD*/ | 18596 WRITE_REGS
              CALL PUT_NEWQUAD;                               /*NEWQUAD*/ | 18608 WRITE_REGS
              END;                                            /*NEWQUAD*/ | 18616 WRITE_REGS
           LL=NEXT_VAR(LL);                                   /*NEWQUAD*/ | 18616 WRITE_REGS
           END;                                               /*NEWQUAD*/ | 18632 WRITE_REGS
        END;                                                  /*NEWQUAD*/ | 18640 WRITE_REGS
    END;                                                      /*NEWQUAD*/ | 18640 WRITE_REGS
  END;                                                        /*NEWQUAD*/ | 18648 WRITE_REGS
```

```
 NEWQUAD_GEN:PROCEDURE;                                              /*NEWQUAD*/ | 18654
                                                                                 | 18654 NEWQUAD_GE
 /*NEWQUAD_GEN  REGISTER QUAD GENERATOR  */                                      | 18654 NEWQUAD_GE
                                                                                 | 18654 NEWQUAD_GE
  DECLARE (REG1,REG2,REG3,PRE_VAR)FIXED;                             /*NEWQUAD*/ | 18654 NEWQUAD_GE
         DECLARE CASE_NUM(30) FIXED INITIAL(0,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            1,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            1,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            1,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            0,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            0,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            3,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            6,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            2,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            2,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            0,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            1,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            1,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            1,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            0,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            0,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            0,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            0,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            5,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            0,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            0,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            0,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            0,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            0,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            4,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            0,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            0,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            0,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            0,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            0,                       /*NEWQUAD*/ | 18666 NEWQUAD_GE
                                            5);                      /*NEWQUAD*/ | 18666 NEWQUAD_GE
  DO CASE CASE_NUM(OPER);                                            /*NEWQUAD*/ | 18666 NEWQUAD_GE
     DO;                                                             /*NEWQUAD*/ | 18698 NEWQUAD_GE
                                                                                 | 18698 NEWQUAD_GE
 /*PROCESS ASIGN QUAD    */                                                      | 18698 NEWQUAD_GE
                                                                                 | 18698 NEWQUAD_GE
         REG1=ALLOC_REG_DP(OPERAND1);                                /*NEWQUAD*/ | 18698 NEWQUAD_GE
         REG2 = FIND_RES(RESULT);                                    /*NEWQUAD*/ | 18718 NEWQUAD_GE
         IF ADDRSS(REG2)=TRUE THEN DO;                               /*NEWQUAD*/ | 18738 NEWQUAD_GE
            OPRTOR=0;                                                /*NEWQUAD*/ | 18760 NEWQUAD_GE
            OPRND1=REG1;                                             /*NEWQUAD*/ | 18766 NEWQUAD_GE
            OPRND2=0;                                                /*NEWQUAD*/ | 18774 NEWQUAD_GE
            RSLT=REG2;                                               /*NEWQUAD*/ | 18780 NEWQUAD_GE
            CALL PUT_NEWQUAD;                                        /*NEWQUAD*/ | 18788 NEWQUAD_GE
            CURR_MIC=CURR_MIC+2;                                     /*NEWQUAD*/ | 18796 NEWQUAD_GE
            ADDRSS(REG2)=FALSE;                                      /*NEWQUAD*/ | 18808 NEWQUAD_GE
            DEALLOCABL(REG2)=FALSE;                                  /*NEWQUAD*/ | 18818 NEWQUAD_GE
            END;                                                     /*NEWQUAD*/ | 18828 NEWQUAD_GE
         ELSE DO;                                                    /*NEWQUAD*/ | 18836 NEWQUAD_GE
            VARNUM(REG1)=VARNUM(REG1)+1;                             /*NEWQUAD*/ | 18836 NEWQUAD_GE
            REFRENCE(REG1)=REFNO;                                    /*NEWQUAD*/ | 18864 NEWQUAD_GE
            REFNO=REFNO+1;                                           /*NEWQUAD*/ | 18880 NEWQUAD_GE
            IF BYTE(RESULT,0)=BYTE('.') THEN                         /*NEWQUAD*/ | 18892 NEWQUAD_GE
               TEMP(REG1)=TEMP(REG1)+1;                              /*NEWQUAD*/ | 18922 NEWQUAD_GE
            LL=LAVS;                                                 /*NEWQUAD*/ | 18942 NEWQUAD_GE
            LAVS=NEXT_VAR(LAVS);                                     /*NEWQUAD*/ | 18950 NEWQUAD_GE
            NEXT_VAR(LL)=POINT(REG1);                                /*NEWQUAD*/ | 18966 NEWQUAD_GE
            POINT(REG1)=LL;                                          /*NEWQUAD*/ | 18990 NEWQUAD_GE
            VARIBLS(LL)=RESULT;                                      /*NEWQUAD*/ | 19006 NEWQUAD_GE
            CHANGE(LL)=TRUE;                                         /*NEWQUAD*/ | 19022 NEWQUAD_GE
            END;                                                     /*NEWQUAD*/ | 19034 NEWQUAD_GE
```

```
          [illegible](REG1)=FALSE;                                   /*NEWQUAD*/  | 19034 NEWQUAD_GE
          CALL DEALLOC_TEMP(OPERAND1,REG1);                          /*NEWQUAD*/  | 19044 NEWQUAD_GE
          IF RD=FALSE THEN DO;                                       /*NEWQUAD*/  | 19068 NEWQUAD_GE
             LL=POINT(REG2);                                         /*NEWQUAD*/  | 19086 NEWQUAD_GE
             NEXT_VAR(LL)=LAVS;                                      /*NEWQUAD*/  | 19102 NEWQUAD_GE
             LAVS=LL;                                                /*NEWQUAD*/  | 19118 NEWQUAD_GE
             STATUS(REG2)=FALSE;                                     /*NEWQUAD*/  | 19126 NEWQUAD_GE
             END;                                                    /*NEWQUAD*/  | 19136 NEWQUAD_GE
          END;                                                       /*NEWQUAD*/  | 19136 NEWQUAD_GE
      DO;                                                            /*NEWQUAD*/  | 19136 NEWQUAD_GE
                                                                                  | 19144 NEWQUAD_GE
 /* PROCESS ADD, SUB, MUL, LT, GT, EQ QUADS      */                               | 19144 NEWQUAD_GE
                                                                                  | 19144 NEWQUAD_GE
          REG1=ALLOC_REG_OP(OPERAND1);                               /*NEWQUAD*/  | 19144 NEWQUAD_GE
          REG2=ALLOC_REG_OP(OPERAND2);                               /*NEWQUAD*/  | 19164 NEWQUAD_GE
          REG3=ALLOC_REG_RES(RESULT);                                /*NEWQUAD*/  | 19184 NEWQUAD_GE
          OPRND1=REG1;                                               /*NEWQUAD*/  | 19204 NEWQUAD_GE
          OPRND2=REG2;                                               /*NEWQUAD*/  | 19212 NEWQUAD_GE
          RSLT=REG3;                                                 /*NEWQUAD*/  | 19220 NEWQUAD_GE
             OPRTOR=OPER;                                            /*NEWQUAD*/  | 19228 NEWQUAD_GE
          CALL PUT_NEWQUAD;                                          /*NEWQUAD*/  | 19236 NEWQUAD_GE
          CALL DEALLOC_TEMP(OPERAND1,REG1);                          /*NEWQUAD*/  | 19244 NEWQUAD_GE
          CALL DEALLOC_TEMP(OPERAND2,REG2);                          /*NEWQUAD*/  | 19268 NEWQUAD_GE
          IF OPER=MUL THEN DO;                                       /*NEWQUAD*/  | 19292 NEWQUAD_GE
             IF CURR_MIC MOD 2 = 1 THEN                              /*NEWQUAD*/  | 19308 NEWQUAD_GE
                CURR_MIC=CURR_MIC+12;                                /*NEWQUAD*/  | 19342 NEWQUAD_GE
             ELSE                                                    /*NEWQUAD*/  | 19346 NEWQUAD_GE
                CURR_MIC=CURR_MIC+13;                                /*NEWQUAD*/  | 19346 NEWQUAD_GE
             END;                                                    /*NEWQUAD*/  | 19366 NEWQUAD_GE
          ELSE                                                       /*NEWQUAD*/  | 19366 NEWQUAD_GE
             CURR_MIC=CURR_MIC+3;                                    /*NEWQUAD*/  | 19366 NEWQUAD_GE
          END;                                                       /*NEWQUAD*/  | 19386 NEWQUAD_GE
      DO;                                                            /*NEWQUAD*/  | 19386 NEWQUAD_GE
                                                                                  | 19394 NEWQUAD_GE
 /*PROCESS BT, BF QUADS       */                                                  | 19394 NEWQUAD_GE
                                                                                  | 19394 NEWQUAD_GE
          REG1=ALLOC_REG_OP(OPERAND1);                               /*NEWQUAD*/  | 19394 NEWQUAD_GE
          CALL DEALLOC_TEMP(OPERAND1,REG1);                          /*NEWQUAD*/  | 19414 NEWQUAD_GE
          CALL WRITE_REGS;                                           /*NEWQUAD*/  | 19438 NEWQUAD_GE
          OPRND1=REG1;                                               /*NEWQUAD*/  | 19446 NEWQUAD_GE
          TEMP_CHAR=RESULT;                                          /*NEWQUAD*/  | 19454 NEWQUAD_GE
          RSLT=LAB_INDEX;                                            /*NEWQUAD*/  | 19462 NEWQUAD_GE
          OPRND2=0;                                                  /*NEWQUAD*/  | 19474 NEWQUAD_GE
             OPRTOR=OPER;                                            /*NEWQUAD*/  | 19480 NEWQUAD_GE
          PRE_VAR= CURR_MIC MOD 2;                                   /*NEWQUAD*/  | 19488 NEWQUAD_GE
          IF PRE_VAR=0 THEN                                          /*NEWQUAD*/  | 19506 NEWQUAD_GE
             CURR_MIC=CURR_MIC+4;                                    /*NEWQUAD*/  | 19530 NEWQUAD_GE
          ELSE                                                       /*NEWQUAD*/  | 19534 NEWQUAD_GE
             CURR_MIC=CURR_MIC+5;                                    /*NEWQUAD*/  | 19534 NEWQUAD_GE
          CALL PUT_NEWQUAD;                                          /*NEWQUAD*/  | 19554 NEWQUAD_GE
          END;                                                       /*NEWQUAD*/  | 19562 NEWQUAD_GE
      DO;                                                            /*NEWQUAD*/  | 19562 NEWQUAD_GE
                                                                                  | 19570 NEWQUAD_GE
 /*PROCESS HALT QUADS     */                                                      | 19570 NEWQUAD_GE
                                                                                  | 19570 NEWQUAD_GE
          OPRND1=0;                                                  /*NEWQUAD*/  | 19570 NEWQUAD_GE
          OPRND2=0;                                                  /*NEWQUAD*/  | 19576 NEWQUAD_GE
          RSLT=0;                                                    /*NEWQUAD*/  | 19582 NEWQUAD_GE
             OPRTOR=OPER;                                            /*NEWQUAD*/  | 19588 NEWQUAD_GE
          CALL PUT_NEWQUAD;                                          /*NEWQUAD*/  | 19596 NEWQUAD_GE
          END;                                                       /*NEWQUAD*/  | 19604 NEWQUAD_GE
      DO;                                                            /*NEWQUAD*/  | 19604 NEWQUAD_GE
                                                                                  | 19612 NEWQUAD_GE
 /*PROCESS LAB QUADS                         */                                   | 19612 NEWQUAD_GE
                                                                                  | 19612 NEWQUAD_GE
          CALL WRITE_REGS;                                           /*NEWQUAD*/  | 19612 NEWQUAD_GE
```

```
       TEMP_CHAR=OPERAND1;                                         /*NEWQUAD*/   [illegible]  NEWQUAD_GE
       OPRND1=LAB_INDEX;                                           /*NEWQUAD*/   19628  NEWQUAD_GE
       OPRND2=0;                                                   /*NEWQUAD*/   19640  NEWQUAD_GE
       RSLT=0;                                                     /*NEWQUAD*/   19646  NEWQUAD_GE
          OPRTOR=OPER;                                             /*NEWQUAD*/   19652  NEWQUAD_GE
       MIC_LOC(OPRND1)=CURR_MIC;                                   /*NEWQUAD*/   19660  NEWQUAD_GE
       CURR_MIC=CURR_MIC+2;                                        /*NEWQUAD*/   19676  NEWQUAD_GE
       CALL PUT_NEWQUAD;                                           /*NEWQUAD*/   19688  NEWQUAD_GE
       DO II=1 TO MAXREG;                                          /*NEWQUAD*/   19696  NEWQUAD_GE
          IF STATUS(II)=TRUE THEN DO;                              /*NEWQUAD*/   19740  NEWQUAD_GE
             STATUS(II)=FALSE;                                     /*NEWQUAD*/   19762  NEWQUAD_GE
             LL=POINT(II);                                         /*NEWQUAD*/   19772  NEWQUAD_GE
             DO JJ=1 TO VARNUM(II);                                /*NEWQUAD*/   19788  NEWQUAD_GE
                PRE_VAR=NEXT_VAR(LL);                              /*NEWQUAD*/   19840  NEWQUAD_GE
                NEXT_VAR(LL)=LAVS;                                 /*NEWQUAD*/   19856  NEWQUAD_GE
                LAVS=LL;                                           /*NEWQUAD*/   19872  NEWQUAD_GE
                LL=PRE_VAR;                                        /*NEWQUAD*/   19880  NEWQUAD_GE
                END;                                               /*NEWQUAD*/   19888  NEWQUAD_GE
             END;                                                  /*NEWQUAD*/   19896  NEWQUAD_GE
          END;                                                     /*NEWQUAD*/   19896  NEWQUAD_GE
       END;                                                        /*NEWQUAD*/   19904  NEWQUAD_GE
    DO;                                                            /*NEWQUAD*/   19904  NEWQUAD_GE
                                                                                 19912  NEWQUAD_GE
/*PROCESS SUBS,SUBL QUADS      */                                                19912  NEWQUAD_GE
                                                                                 19912  NEWQUAD_GE
       REG1=ALLOC_REG_OP(OPERAND1);                                /*NEWQUAD*/   19912  NEWQUAD_GE
       SUBSFLAG=OPER;                                              /*NEWQUAD*/   19932  NEWQUAD_GE
       OPER=0;                                                     /*NEWQUAD*/   19940  NEWQUAD_GE
       REG2=ALLOC_REG_OP(OPERAND2);                                /*NEWQUAD*/   19946  NEWQUAD_GE
       REG3=ALLOC_REG_RES(RESULT);                                 /*NEWQUAD*/   19966  NEWQUAD_GE
       OPRND1=REG2;                                                /*NEWQUAD*/   19986  NEWQUAD_GE
       OPRND2=0;                                                   /*NEWQUAD*/   19994  NEWQUAD_GE
       RSLT=REG3;                                                  /*NEWQUAD*/   20000  NEWQUAD_GE
       OPRTOR=29;                                                  /*NEWQUAD*/   20008  NEWQUAD_GE
       CALL PUT_NEWQUAD;                                           /*NEWQUAD*/   20016  NEWQUAD_GE
       OPRTOR=1;                                                   /*NEWQUAD*/   20024  NEWQUAD_GE
       OPRND1=REG1;                                                /*NEWQUAD*/   20032  NEWQUAD_GE
       OPRND2=REG3;                                                /*NEWQUAD*/   20040  NEWQUAD_GE
       RSLT=REG3;                                                  /*NEWQUAD*/   20048  NEWQUAD_GE
       CALL PUT_NEWQUAD;                                           /*NEWQUAD*/   20056  NEWQUAD_GE
       IF SUBSFLAG=SUBS THEN DO;                                   /*NEWQUAD*/   20064  NEWQUAD_GE
          OPRTOR=28;                                               /*NEWQUAD*/   20080  NEWQUAD_GE
          OPRND1=REG3;                                             /*NEWQUAD*/   20088  NEWQUAD_GE
          OPRND2=0;                                                /*NEWQUAD*/   20096  NEWQUAD_GE
          RSLT=REG3;                                               /*NEWQUAD*/   20102  NEWQUAD_GE
          CALL PUT_NEWQUAD;                                        /*NEWQUAD*/   20110  NEWQUAD_GE
          CURR_MIC=CURR_MIC+7;                                     /*NEWQUAD*/   20119  NEWQUAD_GE
          END;                                                     /*NEWQUAD*/   20130  NEWQUAD_GE
       ELSE DO;                                                    /*NEWQUAD*/   20130  NEWQUAD_GE
          CURR_MIC=CURR_MIC+5;                                     /*NEWQUAD*/   20138  NEWQUAD_GE
          ADDRSS(REG3)=TRUE;                                       /*NEWQUAD*/   20150  NEWQUAD_GE
          DEALLOCABL(REG3)=TRUE;                                   /*NEWQUAD*/   20162  NEWQUAD_GE
          END;                                                     /*NEWQUAD*/   20174  NEWQUAD_GE
       CALL DEALLOC_TEMP(OPERAND2,REG2);                           /*NEWQUAD*/   20174  NEWQUAD_GE
       DEALLOCABL(REG1)=FALSE;                                     /*NEWQUAD*/   20199  NEWQUAD_GE
       END;                                                        /*NEWQUAD*/   20208  NEWQUAD_GE
    DO;                                                            /*NEWQUAD*/   20208  NEWQUAD_GE
                                                                                 20216  NEWQUAD_GE
/*PROCESS BR QUADS             */                                                20216  NEWQUAD_GE
                                                                                 20216  NEWQUAD_GE
       CALL WRITE_REGS;                                            /*NEWQUAD*/   20216  NEWQUAD_G
       OPRND1=0;                                                   /*NEWQUAD*/   20224  NEWQUAD_G
       OPRND2=0;                                                   /*NEWQUAD*/   20230  NEWQUAD_G
       TEMP_CHAR=RESULT;                                           /*NEWQUAD*/   20236  NEWQUAD_G
       RSLT=LAB_INDEX;                                             /*NEWQUAD*/   20244  NEWQUAD_G
          OPRTOR=OPER;                                             /*NEWQUAD*/   20256  NEWQUAD_G
```

```
|         CALL PUT_NEWQUAD;                                              /*NEWQUAD*/ |        NEWQUAD_GEN
|         END;                                                           /*NEWQUAD*/ | 20272 NEWQUAD_GEN
|      END;                                                              /*NEWQUAD*/ | 20272 NEWQUAD_GEN
|  END;                                                                  /*NEWQUAD*/ | 20280 NEWQUAD_GEN
| PROCESS_QUADS:PROCEDURE;                                               /*NEWQUAD*/ | 20286
|                                                                                    | 20286 PROCESS_QUAD
| /*PROCESS_QUADS     PROCESS GENERATED QUADS AND GENERATE REGISTER QUADS */         | 20286 PROCESS_QUAD
|                                                                                    | 20286 PROCESS_QUAD
|  DECLARE MM FIXED;                                                     /*NEWQUAD*/ | 20286 PROCESS_QUAD
|  OUTPUT(1) = '1              QUADS GENERATED';                         /*NEWQUAD*/ | 20298 PROCESS_QUAD
|  DOUBLE_SPACE;                                                         /*NEWQUAD*/ | 20322 PROCESS_QUAD
|  OUTPUT='OPERATOR  OPERAND1/  OPERAND2 RESULT/' ||                     /*NEWQUAD*/ | 20346 PROCESS_QUAD
|        '  ****  RCD_NR  LOC_QUAD';                                     /*NEWQUAD*/ | 20346 PROCESS_QUAD
|  OUTPUT='            CONDITION              LABEL';                    /*NEWQUAD*/ | 20394 PROCESS_QUAD
|  DO  MM = 1 TO NEXTQUAD-1;                                             /*NEWQUAD*/ | 20404 PROCESS_QUAD
|     CALL READQUAD(MM);                                                 /*NEWQUAD*/ | 20452 PROCESS_QUAD
|     CALL NEWQUAD_GEN;                                                  /*NEWQUAD*/ | 20468 PROCESS_QUAD
|     END;                                                               /*NEWQUAD*/ | 20476 PROCESS_QUAD
|  CALL WRITE_REGS;                                                      /*NEWQUAD*/ | 20484 PROCESS_QUAD
|  END;                                                                  /*NEWQUAD*/ | 20492 PROCESS_QUAD
|     /*     DEBUG PRINT OF SYMBOL TABLE      */                                     | 20498
|                                                                                    | 20498
| PRINTSYMB:                                                                         | 20498
|  PROCEDURE;                                                                        | 20498
|       OUTPUT(1)='1SYMBOL TABLE';                                                   | 20498 PRINTSYMB
|     DOUBLE_SPACE;                                                                  | 20534 PRINTSYMB
|     OUTPUT='INDEX  SYMBOL  LOCATION  DEFINED  SIZE  INITIAL';                      | 20558 PRINTSYMB
|       DO I=1 TO NSYMBOL;                                                           | 20578 PRINTSYMB
|     BUFFER=PAD('    '||I,9);                                                       | 20622 PRINTSYMB
|     BUFFER=PAD(BUFFER||SYMB(I),19);                                                | 20670 PRINTSYMB
|     BUFFER=PAD(BUFFER||LOCAT(I),29);                                               | 20718 PRINTSYMB
|     BUFFER=PAD(BUFFER||DEF(I),36);                                                 | 20774 PRINTSYMB
|     BUFFER=PAD(BUFFER||SIZE(I),43);                                                | 20830 PRINTSYMB
|     BUFFER=PAD(BUFFER||INIT(I),80);                                                | 20886 PRINTSYMB
|     OUTPUT=BUFFER;                                                                 | 20942 PRINTSYMB
|     END;                                                                           | 20962 PRINTSYMB
|     OUTPUT(1)='1CONSTANT TABLE';                                                   | 20970 PRINTSYMB
|     DOUBLE_SPACE;                                                                  | 20994 PRINTSYMB
|     OUTPUT='INDEX  VALUE';                                                         | 21018 PRINTSYMB
|        DO I=1 TO NCONSTANT;                                                        | 21038 PRINTSYMB
|     BUFFER=PAD('    '||I,9);                                                       | 21082 PRINTSYMB
|     BUFFER=PAD(BUFFER||CONVAL(I),80);                                              | 21130 PRINTSYMB
|     OUTPUT=BUFFER;                                                                 | 21186 PRINTSYMB
|        END;                                                                        | 21206 PRINTSYMB
|   OUTPUT(1)='1LABEL TABLE';                                                        | 21214 PRINTSYMB
|     DOUBLE_SPACE;                                                                  | 21238 PRINTSYMB
|     OUTPUT='INDEX  LABEL  ADDRESS  DEFINED';                                       | 21262 PRINTSYMB
|     DO I=1 TO NLABEL;                                                              | 21282 PRINTSYMB
|     BUFFER=PAD('    '||I,9);                                                       | 21326 PRINTSYMB
|     BUFFER=PAD(BUFFER||LABID(I),16);                                               | 21374 PRINTSYMB
|     BUFFER=PAD(BUFFER||LASTREF(I),27);                                             | 21422 PRINTSYMB
|     BUFFER=PAD(BUFFER||LABDEF(I),80);                                              | 21478 PRINTSYMB
|     OUTPUT=BUFFER;                                                                 | 21534 PRINTSYMB
|     END;                                                                           | 21554 PRINTSYMB
|  END PRINTSYMB;                                                                    | 21562 PRINTSYMB
|                                                                                    | 21568
|     /*     STORE AN ENTIRE QUAD IN THE QUAD TABLE      */                          | 21568
|                                                                                    | 21568
| STOREQUAD :                                                                        | 21568
|  PROCEDURE (QUADNO,OPER,OPND1,OPND2,RES);                                          | 21568
|     DECLARE (QUADNO,OPER,OPND1,OPND2,RES) FIXED;                                   | 21568 STOREQUAD
|     RCD_NR=(QUADNO-1)/MAXQUADS;                                                    | 21580 STOREQUAD
|     IF RCD_NR ¬= RCD_BUFF THEN                                                     | 21602 STOREQUAD
|          DO;                                                                       | 21626 STOREQUAD
|             FILE(1,RCD_BUFF)=QUADS;                                                | 21618 STOREQUAD
|             RCD_BUFF=RCD_BUFF+1;                                                   | 21640 STOREQUAD
```

```
            END;                                                      21652 STOREQUA
    LOC_QUAD=((QUADNO-1) MOD MAXQUADS)*4+1;                           21652 STOREQUA
    QUADS (LOC_QUAD  )=OPER;                                          21684 STOREQUA
    QUADS (LOC_QUAD+1)=OPND1;                                         21700 STOREQUA
    QUADS (LOC_QUAD+2)=OPND2;                                         21720 STOREQUA
    QUADS (LOC_QUAD+3)=RES;                                           21740 STOREQUA
    RETURN;                                                           21760 STOREQUA
 END STOREQUAD;                                                       21765 STOREQUA
                                                                      21772
    /*      GET QUAD - RETRIEV A FIELD FROM A QUAD    */              21772
                                                                      21772
GETQUAD:                                                              21772
  PROCEDURE (QUADNO,INDEX) FIXED;                                     21772
    DECLARE (QUADNO,INDEX) FIXED;                                     21772 GETQUAD
    RCD_NR=(QUADNO-1)/MAXQUADS;                                       21784 GETQUAD
    IF RCD_NR ¬= RCD_BUFF THEN                                        21806 GETQUAD
          DO;                                                         21830 GETQUAD
            FILE(1,RCD_BUFF)=QUADS;                                   21822 GETQUAD
            RCD_BUFF=RCD_NR;                                          21844 GETQUAD
          QUADS=FILE(1,RCD_BUFF);                                     21852 GETQUAD
          END;                                                        21882 GETQUAD
    LOC_QUAD=((QUADNO-1) MOD MAXQUADS)*4+1;                           21832 GETQUAD
    RETURN QUADS(LOC_QUAD+INDEX-1);                                   21914 GETQUAD
END GETQUAD;                                                          21942 GETQUAD
                                                                      21948
    /*      PUT A FIELD IN A QUAD     */                              21948
                                                                      21948
PUTQUAD:                                                              21948
 PROCEDURE (QUADNO,INDEX,VALUE);                                      21948
    DECLARE (QUADNO,INDEX,VALUE) FIXED;                               21948 PUTQUAD
    RCD_NR=(QUADNO-1)/MAXQUADS;                                       21960 PUTQUAD
    IF RCD_NR ¬= RCD_BUFF THEN                                        21982 PUTQUAD
          DO;                                                         22006 PUTQUAD
            FILE(1,RCD_BUFF)=QUADS;                                   21998 PUTQUAD
            RCD_BUFF=RCD_NR;                                          22020 PUTQUAD
          QUADS=FILE(1,RCD_BUFF);                                     22028 PUTQUAD
          END;                                                        22058 PUTQUAD
    LOC_QUAD=((QUADNO-1) MOD MAXQUADS)*4+1;                           22058 PUTQUAD
    QUADS(LOC_QUAD+INDEX-1)=VALUE;                                    22090 PUTQUAD
    RETURN;                                                           22114 PUTQUAD
 END PUTQUAD;                                                         22120 PUTQUAD
                                                                      22126
    /* PUT A TEMPORARY VARIABLE IN THE SYMBOL TABLE. THESE            22126
           ARE GENERATED BY THE PROGRAM AND ARE ALL UNIQUE. THE       22126
           ONLY ERROR IS OVERFLOW          */                         22126
                                                                      22126
PUTTEMP:                                                              22126
    PROCEDURE (STACKLOC);                                             22126
          DECLARE STACKLOC FIXED; /* LOCATION OF TEMP IN STACK */     22126 PUTTEMP
          IF NSYMBOL = SYMBOLS THEN                                   22138 PUTTEMP
            DO;                                                       22162 PUTTEMP
               OUTPUT='*** SYMBOL TABLE OVERFLOW, MAX IS '||SYMBOLS;  22154 PUTTEMP
               CALL EXIT;                                             22200 PUTTEMP
            END;                                                      22214 PUTTEMP
          NSYMBOL=NSYMBOL+1;                                          22214 PUTTEMP
          TABLE_LOC(STACKLOC)=NSYMBOL;                                22226 PUTTEMP
          SIZE (NSYMBOL)=0;                                           22242 PUTTEMP
          DEF  (NSYMBOL)=1;                                           22256 PUTTEMP
          LOCAT(NSYMBOL)=0;                                           22272 PUTTEMP
          INIT (NSYMBOL)=0;                                           22286 PUTTEMP
          SYMB (NSYMBOL)=VAR(STACKLOC);                               22300 PUTTEMP
    RETURN;                                                           22324 PUTTEMP
END PUTTEMP;                                                          22330 PUTTEMP
                                                                      22336
                                                                      22336
    /*  FIND LABEL IN LABEL TABLE. ERROR IF NOT FOUND */              22336
```

```
                                                              |
    FINDLAB:                                                  | 22336
                                                              | 22336
      PROCEDURE;                                              | 22336
    DECLARE LAB CHARACTER;                                    | 22336 FINDLAB
       LAB = VAR(SP);                                         | 22348 FINDLAB
      I = 1;                                                  | 22364 FINDLAB
        DO WHILE I <= NLABEL;                                 | 22372 FINDLAB
          IF LABID(I) = LAB THEN DO;                          | 22388 FINDLAB
            TABLE_LOC(SP) = I;                                | 22438 FINDLAB
          RETURN;                                             | 22454 FINDLAB
        END;                                                  | 22460 FINDLAB
        I = I+1;                                              | 22460 FINDLAB
      END;                                                    | 22472 FINDLAB
        IF NLABEL < LABELS THEN DO;                           | 22480 FINDLAB
        NLABEL = NLABEL + 1;                                  | 22496 FINDLAB
        LABID(NLABEL) = LAB;                                  | 22508 FINDLAB
        LABDEF(NLABEL)=0;                                     | 22524 FINDLAB
      TABLE_LOC(SP) = NLABEL;                                 | 22538 FINDLAB
      RETURN;                                                 | 22554 FINDLAB
      END;                                                    | 22560 FINDLAB
    ELSE DO;                                                  | 22560 FINDLAB
      OUTPUT='*** LABEL TABLE OVERFLOW, MAX IS '||LABELS;     | 22568 FINDLAB
        CALL EXIT;                                            | 22614 FINDLAB
      END;                                                    | 22628 FINDLAB
    END FINDLAB;                                              | 22628 FINDLAB
                                                              | 22634
     /* INSERTS A BRANCH LABEL INTO THE LABEL TABLE  */       | 22634
                                                              | 22634
  BRANCHLAB:                                                  | 22634
    PROCEDURE(LAB);                                           | 22634
    DECLARE LAB CHARACTER;                                    | 22634 BRANCHLA
      NLABEL=NLABEL+1;                                        | 22640 BRANCHLA
        IF NLABEL=LABELS THEN DO;                             | 22658 BRANCHLA
        OUTPUT='*** LABEL TABLE OVERFLOW';                    | 22674 BRANCHLA
        CALL EXIT;                                            | 22694 BRANCHLA
      END;                                                    | 22708 BRANCHLA
      LABID(NLABEL)=LAB;                                      | 22708 BRANCHLA
      LABDEF(NLABEL)=0;                                       | 22724 BRANCHLA
      VAR(SP-1)=LAB;                                          | 22738 BRANCHLA
      TABLE_LOC(SP-1)=NLABEL;                                 | 22758 BRANCHLA
      SAVEINDEX=NLABEL;                                       | 22778 BRANCHLA
      RETURN;                                                 | 22786 BRANCHLA
  END BRANCHLAB;                                              | 22792 BRANCHLA
                                                              | 22798
                                                              | 22798
QUADGEN:                                                      | 22798
    PROCEDURE (TYPE);                                         | 22798
                                                              | 22798 QUADGEN
/*   DUMMY QUADGEN   */                                       | 22798 QUADGEN
                                                              | 22798 QUADGEN
          DECLARE SLOC FIXED;                                 | 22798 QUADGEN
          DECLARE TYPE FIXED; /* QUAD TYPE */                 | 22810 QUADGEN
                                                              | 22810 QUADGEN
          DECLARE (TNAME,OPND1,OPND2) CHARACTER;              | 22810 QUADGEN
  /*   OPERANDS FOR STOREQUAD      */                         | 22810 QUADGEN
    DECLARE (ITYP,IOP1,IOP2,IRES) FIXED;                      | 22810 QUADGEN
    ITYP=TYPE;                                                | 22810 QUADGEN
    IOP1=0;                                                   | 22818 QUADGEN
    IOP2=0;                                                   | 22824 QUADGEN
    IRES=0;                                                   | 22830 QUADGEN
                                                              | 22836 QUADGEN
          IF TYPE <= MMOD THEN DO;                            | 22836 QUADGEN
    I=TABLE_LOC(SP-2);                                        | 22852 QUADGEN
    IOP1=I;                                                   | 22872 QUADGEN
     IF  I < 0 THEN                                           | 22880 QUADGEN
```

```
              OPND1=CONVAL(-I);                                   22904 QUADGEN
           ELSE                                                   22922 QUADGEN
              OPND1=SYMB(I);                                      22922 QUADGEN
   I=TABLE_LOC(SP);                                               22946 QUADGEN
   IOP2=I;                                                        22952 QUADGEN
    IF  I < 0 THEN                                                22970 QUADGEN
              OPND2=CONVAL(-I);                                   22994 QUADGEN
           ELSE                                                   23012 QUADGEN
              OPND2=SYMB(I);                                      23012 QUADGEN
              TNAME='.T'||NEXTQUAD;                               23036 QUADGEN
          VAR(SP-2)=TNAME;                                        23068 QUADGEN
          CALL  PUTTEMP(SP-2);                                    23088 QUADGEN
   IRES=TABLE_LOC(SP-2);                                          23108 QUADGEN
          END;                                                    23128 QUADGEN
   ELSE IF TYPE = HALT THEN                                       23128 QUADGEN
   DO;                                                            23160 QUADGEN
   END;                                                           23152 QUADGEN
   ELSE IF TYPE <= BF THEN                                        23152 QUADGEN
     DO;                                                          23164 QUADGEN
    IRES=TABLE_LOC(SP);                                           23176 QUADGEN
    IOP1=0;                                                       23192 QUADGEN
    IOP2=0;                                                       23198 QUADGEN
   OPND1=0;                                                       23204 QUADGEN
   OPND2=0;                                                       23218 QUADGEN
     END;                                                         23232 QUADGEN
   ELSE IF TYPE = REL THEN DO;                                    23232 QUADGEN
         TNAME='.T'||NEXTQUAD;                                    23256 QUADGEN
   I=TABLE_LOC(SP-2);                                             23258 QUADGEN
   IOP1=I;                                                        23308 QUADGEN
    IF  I < 0 THEN                                                23316 QUADGEN
              OPND1=CONVAL(-I);                                   23340 QUADGEN
           ELSE                                                   23358 QUADGEN
              OPND1=SYMB(I);                                      23358 QUADGEN
   I=TABLE_LOC(SP);                                               23382 QUADGEN
   IOP2=I;                                                        23398 QUADGEN
    IF  I < 0 THEN                                                23406 QUADGEN
              OPND2=CONVAL(-I);                                   23430 QUADGEN
           ELSE                                                   23448 QUADGEN
              OPND2=SYMB(I);                                      23448 QUADGEN
          VAR(SP-2)=TNAME;                                        23472 QUADGEN
   CALL  PUTTEMP(SP-2);                                           23492 QUADGEN
   IRES=TABLE_LOC(SP-2);                                          23512 QUADGEN
   ITYP=FIXV(SP-1);                                               23532 QUADGEN
          END;                                                    23552 QUADGEN
   ELSE IF TYPE = ASGN THEN                                       23552 QUADGEN
          DO;                                                     23584 QUADGEN
   I=TABLE_LOC(SP);                                               23576 QUADGEN
   IOP1=I;                                                        23592 QUADGEN
    IF  I < 0 THEN                                                23600 QUADGE
              OPND1=CONVAL(-I);                                   23624 QUADGE
           ELSE                                                   23642 QUADGE
              OPND1=SYMB(I);                                      23642 QUADGE
   IRES=TABLE_LOC(SP-2);                                          23666 QUADGE
     END;                                                         23686 QUADGE
   ELSE IF TYPE = SUBS THEN                                       23686 QUADGE
          DO;                                                     23718 QUADGE
              TNAME='.T'||NEXTQUAD;                               23710 QUADGE
              I=TABLE_LOC(SP-1);                                  23742 QUADGE
   IOP2=I;                                                        23762 QUADGE
              IF I<0 THEN                                         23770 QUADGE
                 OPND2=CONVAL(-I);                                23794 QUADGE
          ELSE                                                    23812 QUADGE
                 OPND2=SYMB(I);                                   23812 QUADGE
          SLOC=TABLE_LOC(SP-2);                                   23836 QUADGE
          IOP1=SLOC;                                              23856 QUADGE
          OPND1=SYMB(SLOC);                                       23864 QUADGE
```

```
            VAR(SP-2)=TNAME;                                        23880 QUADGEN
            CALL PUTTEMP(SP-2);                                     23900 QUADGEN
      IRES=TABLE_LOC(SP-2);                                         23920 QUADGEN
            LOCAT(NSYMBOL)=-SLOC;                                   23940 QUADGEN
            END;                                                    23958 QUADGEN
    ELSE IF TYPE = BZ THEN DO;                                      23958 QUADGEN
      IRES = TABLE_LOC(SP-1);                                       23982 QUADGEN
      IOP1=0;                                                       24002 QUADGEN
      IOP2=0;                                                       24008 QUADGEN
      OPND1=0;                                                      24014 QUADGEN
      OPND2=0;                                                      24028 QUADGEN
   END;                                                             24042 QUADGEN
    ELSE IF TYPE <= OR THEN DO;                                     24042 QUADGEN
      TNAME = '.T'||NEXTQUAD;                                       24066 QUADGEN
     I = TABLE_LOC(SP-2);                                           24098 QUADGEN
    IOP1 = I;                                                       24118 QUADGEN
    IF I < 0 THEN                                                   24126 QUADGEN
      OPND1 = CONVAL(-I);                                           24150 QUADGEN
    ELSE                                                            24168 QUADGEN
      OPND1 = SYMB(I);                                              24168 QUADGEN
    I = TABLE_LOC(SP);                                              24192 QUADGEN
    IOP2 = I;                                                       24208 QUADGEN
    IF I < 0 THEN                                                   24216 QUADGEN
      OPND2 = CONVAL(-I);                                           24240 QUADGEN
    ELSE                                                            24258 QUADGEN
      OPND2 = SYMB(I);                                              24258 QUADGEN
    VAR(SP-2)=TNAME;                                                24282 QUADGEN
    CALL PUTTEMP(SP-2);                                             24302 QUADGEN
   IRES=TABLE_LOC(SP-2);                                            24322 QUADGEN
   END;                                                             24342 QUADGEN
    ELSE IF TYPE = UMIN THEN                                        24342 QUADGEN
      DO:                                                           24374 QUADGEN
     I=TABLE_LOC(SP);                                               24366 QUADGEN
      IOP1=I;                                                       24392 QUADGEN
      IOP2=0;                                                       24390 QUADGEN
     IF I < 0 THEN                                                  24396 QUADGEN
        OPND1=CONVAL(-I);                                           24420 QUADGEN
      ELSE                                                          24438 QUADGEN
        OPND1=SYMB(I);                                              24436 QUADGEN
      OPND2='----';                                                 24462 QUADGEN
      TNAME = '.T'||NEXTQUAD;                                       24470 QUADGEN
      VAR(SP)=TNAME;                                                24502 QUADGEN
      CALL PUTTEMP(SP);                                             24518 QUADGEN
      IRES=TABLE_LOC(SP);                                           24534 QUADGEN
   END;                                                             24550 QUADGEN
    ELSE IF TYPE = ZQ THEN DO;                                      24550 QUADGEN
      I=TABLE_LOC(SP-1);                                            24574 QUADGEN
        IOP1=I;                                                     24594 QUADGEN
        OPND1=I;                                                    24602 QUADGEN
      IOP2=0;                                                       24618 QUADGEN
      OPND2=0;                                                      24624 QUADGEN
      TNAME=' ';                                                    24638 QUADGEN
      IRES=0;                                                       24646 QUADGEN
    END;                                                            24652 QUADGEN
     ELSE IF TYPE = LAB THEN                                        24652 QUADGEN
            DO;                                                     24684 QUADGEN
              IOP1=TABLE_LOC(SP-1);                                 24676 QUADGEN
          OPND1=VAR(SP-1);                                          24696 QUADGEN
      END;                                                          24716 QUADGEN
       CALL STOREQUAD (NEXTQUAD,ITYP,IOP1,IOP2,IRES);               24716 QUADGEN
     NEXTQUAD=NEXTQUAD+1;                                           24764 QUADGEN
  END QUADGEN;                                                      24776 QUADGEN
                                                                    24782
                                                                    24782
                                                                    24782
      /*  INITIALLY DEFINES A NEW LABEL OR DEFINES A LABEL PREVIOUSLY     24782
```

```
|          STORED  */                                                        | 24782
|                                                                            | 24782
|    DEFINELAB:                                                              | 24782
|                                                                            | 24782
|       PROCEDURE(STACKLOC);                                                 | 24782
|    DECLARE LAB CHARACTER;                                                  | 24782 DEFINELA
|   DECLARE TEMPREF FIXED;                                                   | 24794 DEFINELA
|       DECLARE STACKLOC FIXED; /*  LOC IN STACK OF LABEL */                 | 24794 DEFINELA
|          TABLE_LOC(STACKLOC) = 0;                                          | 24794 DEFINELA
|          LAB = VAR(STACKLOC);                                              | 24808 DEFINELA
|             I = 1;                                                         | 24824 DEFINELA
|               DO WHILE I <= NLABEL;                                        | 24832 DEFINELA
|                  IF LABID(I) = LAB THEN DO;                                | 24848 DEFINELA
|                       LABDEF(I) = 1;                                       | 24898 DEFINELA
|                     TABLE_LOC(STACKLOC) = I;                               | 24914 DEFINELA
|               RETURN;                                                      | 24930 DEFINELA
|              END;                                                          | 24936 DEFINELA
|             I = I+1;                                                       | 24936 DEFINELA
|          END;                                                              | 24948 DEFINELA
|                  NLABEL = NLABEL + 1;                                      | 24956 DEFINELA
|                     IF NLABEL = LABELS THEN DO;                            | 24968 DEFINELA
|                        OUTPUT = ' *** LABEL TABLE OVERFLOW';               | 24984 DEFINELA
|                        CALL EXIT;                                          | 25004 DEFINELA
|                     END;                                                   | 25018 DEFINELA
|                     LABID(NLABEL)= LAB;                                    | 25018 DEFINELA
|                     LABDEF(NLABEL) = 1;                                    | 25034 DEFINELA
|                     TABLE_LOC(STACKLOC) = NLABEL;                          | 25050 DEFINELA
|              RETURN;                                                       | 25066 DEFINELA
|    END DEFINELAB;                                                          | 25072 DEFINELA
|               VAR(SP-1)=LAB;                                               | 25078
|   /* SET THE SIZE COLUMN OF THE SYMBOL TABLE TO THE ARRAY DIMENSION        | 25114
|                  OF DIMENSIONED VARIABLES   */                             | 25114
|                                                                            | 25114
| SETDIM:                                                                    | 25114
|    PROCEDURE;                                                              | 25114
|          K=FIXV(SP-2);                                                     | 25114 SETDIM
|          J=NSYMBOL;                                                        | 25146 SETDIM
|          DO I=1 TO NVARDEF;                                                | 25154 SETDIM
|             SIZE(J)=K;                                                     | 25198 SETDIM
|             J=J-1;                                                         | 25214 SETDIM
|          END;                                                              | 25226 SETDIM
|    END SETDIM;                                                             | 25234 SETDIM
|                                                                            | 25240
|    /* STORE A SYMBOL IN THE SYMBOL TABLE. ERROR IF IT IS ALREADY           | 25240
|               DEFINED. SET ALL DEFAULTS IN THE TABLE      */               | 25240
|                                                                            | 25240
| PUTIT:                                                                     | 25240
|    PROCEDURE (STACKLOC);                                                   | 25240
|       DECLARE STACKLOC FIXED; /* LOC IN STACK OF VARIABLE IDENT */         | 25240 PUTIT
|          IF NSYMBOL=SYMBOLS THEN                                           | 25252 PUTIT
|            DO;                                                             | 25276 PUTIT
|                OUTPUT='*** SYMBOL TABLE OVERFLOW, MAX IS '||SYMBOLS;       | 25268 PUTIT
|                CALL EXIT;                                                  | 25314 PUTIT
|            END;                                                            | 25328 PUTIT
|          TABLE_LOC(STACKLOC)=0;                                            | 25328 PUTIT
|          SYMB(NSYMBOL+1)=VAR(STACKLOC);                                    | 25342 PUTIT
|          I=1;                                                              | 25370 PUTIT
|          S=VAR(STACKLOC);                                                  | 25378 PUTIT
|            DO WHILE SYMB(I) ¬= S;                                          | 25394 PUTIT
|               I=I+1;                                                       | 25444 PUTIT
|            END;                                                            | 25456 PUTIT
|          IF I <= NSYMBOL THEN                                              | 25464 PUTIT
|            DO;                                                             | 25498 PUTIT
|                OUTPUT='DUPLICATE IDENTIFIER (VARIABLE) NAME '||S;          | 25480 PUTIT
|                RETURN;                                                     | 25518 PUTIT
|            END;                                                            | 25524 PUTIT
```

```
         NSYMBOL=1;                                                          25524 PUTIT
         TABLE_LOC(STACKLOC)=NSYMBOL;                                        25532 PUTIT
         SIZE(NSYMBOL)=1;                                                    25548 PUTIT
         LOCAT(NSYMBOL)=0;                                                   25564 PUTIT
         DEF(NSYMBOL)=0;                                                     25578 PUTIT
         INIT(NSYMBOL)=0;                                                    25592 PUTIT
    RETURN;                                                                  25606 PUTIT
END PUTIT;                                                                   25612 PUTIT
                                                                             25618
    /* FIND AN IDENTIFIER IN THE SYMBOL TABLE. ERROR IF NOT FOUND */         25618
                                                                             25618
FINDIT:                                                                      25618
    PROCEDURE (PS);                                                          25618
         DECLARE PS FIXED; /* LOCATION IN STACK OF SYMBOL */                 25618 FINDIT
         S=VAR(PS);                                                          25630 FINDIT
         I=1;                                                                25646 FINDIT
         DO WHILE I <= NSYMBOL;                                              25654 FINDIT
           IF SYMB(I) = S THEN                                               25670 FINDIT
              DO;                                                            25728 FINDIT
                 TABLE_LOC(PS)=I;                                            25720 FINDIT
                 RETURN;                                                     25736 FINDIT
              END;                                                           25742 FINDIT
            I=I+1;                                                           25742 FINDIT
           END;                                                              25754 FINDIT
         OUTPUT='*** VARIABLE IDENTIFIER NOT FOUND - '||S;                   25762 FINDIT
         TABLE_LOC(PS)=0;                                                    25800 FINDIT
    RETURN;                                                                  25814 FINDIT
END FINDIT;                                                                  25820 FINDIT
                                                                             25826
    /* FIND A NUMBER IN THE CONSTANT TABLE. IF NOT FOUND, ADD IT */          25826
                                                                             25826
FINDNO:                                                                      25826
    PROCEDURE;                                                               25826
         L=FIXV(SP);                                                         25826 FINDNO
         CONVAL(NCONSTANT+1)=L;                                              25854 FINDNO
         CONLOC(NCONSTANT)=0;                                                25874 FINDNO
         I=1;                                                                25888 FINDNO
         DO WHILE CONVAL(I) ¬= L;                                            25896 FINDNO
              I=I+1;                                                         25920 FINDNO
         END;                                                                25932 FINDNO
         TABLE_LOC(SP)=-I;                                                   25940 FINDNO
      IF I <= NCONSTANT THEN RETURN;                                         25958 FINDNO
         IF NCONSTANT+1 = CONSTANTS THEN                                     25980 FINDNO
           DO;                                                               26008 FINDNO
              L=CONSTANTS-1;                                                 26000 FINDNO
              OUTPUT='*** CONSTANT TABLE OVERFLOW, MAX IS '||L;              26012 FINDNO
              CALL EXIT;                                                     26058 FINDNO
         END;                                                                26072 FINDNO
         NCONSTANT=I;                                                        26072 FINDNO
    RETURN;                                                                  26080 FINDNO
END FINDNO;                                                                  26086 FINDNO
                                                                             26092
                                                                             26092
    /* MOVE ALL STACK CONTENTS FROM F TO T */                                26092
                                                                             26092
MOVESTACK:                                                                   26092
    PROCEDURE (F,T);                                                         26092
         DECLARE (F,T) FIXED;                                                26092 MOVESTAC
           TABLE_LOC(T)=TABLE_LOC(F);                                        26104 MOVESTAC
           VAR(T)=VAR(F);                                                    26128 MOVESTAC
           FIXV(T)=FIXV(F);                                                  26152 MOVESTAC
         PARSE_STACK(T)=PARSE_STACK(F);                                      26176 MOVESTAC
    END MOVESTACK;                                                           26194 MOVESTAC
                                                                             26200
                                                                             26200
                                                                             26200
```

```
  /* GENLOOP GENERATES THE QUADS NECESSARY FOR DO STATEMENTS */                   | 26200
                                                                                  | 26200
GENLOOP:                                                                          | 26200
  PROCEDURE;                                                                      | 26200
    SP=SP-1;                                                                      | 26200 GENLOOP
    CALL QUADGEN(ASGN);                                                           | 26224 GENLOOP
    LOOP_INDEX=TABLE_LOC(SP-2);                                                   | 26240 GENLOOP
    SP=SP+1;                                                                      | 26260 GENLOOP
    SAVEVAR=VAR(SP-1);                                                            | 26272 GENLOOP
    SAVELOC=TABLE_LOC(SP-1);                                                      | 26292 GENLOOP
    CALL BRANCHLAB('.L'||NEXTQUAD);                                               | 26312 GENLOOP
    SAVLABNO=SAVLABNO+1;                                                          | 26352 GENLOOP
    SAVLAB(SAVLABNO)=VAR(SP-1);                                                   | 26364 GENLOOP
    CALL DEFINELAB(SP-1);                                                         | 26392 GENLOOP
    CALL QUADGEN(LAB);                                                            | 26412 GENLOOP
    TABLE_LOC(SP)=LOOPLIM;                                                        | 26428 GENLOOP
    TABLE_LOC(SP-2)=LOOP_INDEX;                                                   | 26444 GENLOOP
    FIXV(SP-1)=CT;                                                                | 26464 GENLOOP
    CALL QUADGEN(REL);                                                            | 26484 GENLOOP
    SAVQUDNO=SAVQUDNO+1;                                                          | 26500 GENLOOP
    SAVQUD(SAVQUDNO)=NEXTQUAD;                                                    | 26512 GENLOOP
    CALL QUADGEN(ZO);                                                             | 26528 GENLOOP
    CALL PUTQUAD(NEXTQUAD-1,1,BT);                                                | 26544 GENLOOP
    I=TABLE_LOC(SP-2);                                                            | 26580 GENLOOP
    CALL PUTQUAD(NEXTQUAD-1,2,I);                                                 | 26600 GENLOOP
  RETURN;                                                                         | 26636 GENLOOP
 END GENLOOP;                                                                     | 26642 GENLOOP
                                                                                  | 26648
                                                                                  | 26648
                                                                                  | 26648
 /*                    THE SYNTHESIS ALGORITHM FOR XPL        */                  | 26648
                                                                                  | 26648
                                                                                  | 26648
SYNTHESIZE:                                                                       | 26648
PROCEDURE(PRODUCTION_NUMBER);                                                     | 26648
  DECLARE PRODUCTION_NUMBER FIXED;                                                | 26648 SYNTHESIZE
                                                                                  | 26660 SYNTHESIZE
  /*  THIS PROCEDURE IS RESPONSIBLE FOR THE SEMANTICS (CODE SYNTHESIS), IF        | 26660 SYNTHESIZE
     ANY, OF THE SKELETON COMPILER.  ITS ARGUMENT IS THE NUMBER OF THE            | 26660 SYNTHESIZE
     PRODUCTION WHICH WILL BE APPLIED IN THE PENDING REDUCTION.  THE GLOBAL       | 26660 SYNTHESIZE
     VARIABLES MP AND SP POINT TO THE BOUNDS IN THE STACKS OF THE RIGHT PART      | 26660 SYNTHESIZE
     OF THIS PRODUCTION.                                                          | 26660 SYNTHESIZE
      NORMALLY, THIS PROCEDURE WILL TAKE THE FORM OF A GIANT CASE STATEMENT       | 26660 SYNTHESIZE
     ON PRODUCTION-NUMBER.  HOWEVER, THE SYNTAX CHECKER HAS SEMANTICS (THE        | 26660 SYNTHESIZE
     TERMINATION OF CHECKING) ONLY FOR PRODUCTION 1.                   */        | 26660 SYNTHESIZE
                                                                                  | 26660 SYNTHESIZE
                                                                                  | 26660 SYNTHESIZE
                                                                                  | 26660 SYNTHESIZE
 /*  ONE STATEMENT FOR EACH PRODUCTION OF THE GRAMMER  */                         | 26660 SYNTHESIZE
                                                                                  | 26660 SYNTHESIZE
                                                                                  | 26660 SYNTHESIZE
DO CASE PRODUCTION_NUMBER;                                                        | 26660 SYNTHESIZE
     ;           /* DUMMY CASE SINCE PRODUCTIONS NUMBERED FROM 1 */               | 26684 SYNTHESIZE
 /*  <PROGRAM> ::= <STATEMENT LIST>    */                                         | 26684 SYNTHESIZE
 /*  <PROGRAM>  ::=  <STATEMENT LIST>    */                                       | 26684 SYNTHESIZE
   DO;                                                                            | 26684 SYNTHESIZE
     IF MP ¬= 2 THEN  /* WE DIDN'T GET HERE LEGITIMATELY  */                      | 26692 SYNTHESIZE
        DO;                                                                       | 26716 SYNTHESIZE
           CALL ERROR ('EOF AT INVALID POINT', 1);                                | 26708 SYNTHESIZE
           CALL STACK_DUMP;                                                       | 26728 SYNTHESIZE
   END;                                                                           | 26736 SYNTHESIZE
      COMPILING = FALSE;                                                          | 26736 SYNTHESIZE
         END;                                                                     | 26742 SYNTHESIZE
 /*  <STATEMENT LIST> ::= <STATEMENT>    */                                       | 26742 SYNTHESIZE
                                                                                  | 26742 SYNTHESIZE
 /*  <STATEMENT LIST> ::= <STATEMENT LIST> <STATEMENT>    */                      | 26750 SYNTHESIZE
```

```
|      ;                                                                        | 26758 SYNTHESIZ
| /*   <STATEMENT> ::= <BASIC STATEMENT>     */                                 | 26758 SYNTHESIZ
|      ;                                                                        | 26758 SYNTHESIZ
| /*   <STATEMENT> ::= <IF STATEMENT>     */                                    | 26766 SYNTHESIZ
|      ;                                                                        | 26766 SYNTHESIZ
| /*   <BASIC STATEMENT> ::= <ASSIGNMENT> ;     */                              | 26774 SYNTHESIZ
|    DO;                                                                        | 26774 SYNTHESIZ
|      TBASIC = 1;                                                              | 26782 SYNTHESIZ
|    END;                                                                       | 26790 SYNTHESIZ
| /*   <BASIC STATEMENT> ::= <GROUP> ;     */                                   | 26790 SYNTHESIZ
|      ;                                                                        | 26790 SYNTHESIZ
| /*   <BASIC STATEMENT> ::= <PROCEDURE DEFINITION> ;     */                    | 26798 SYNTHESIZ
|      ;                                                                        | 26798 SYNTHESIZ
| /*   <BASIC STATEMENT> ::= <RETURN STATEMENT> ;     */                        | 26806 SYNTHESIZ
|      ;                                                                        | 26806 SYNTHESIZ
| /*   <BASIC STATEMENT> ::= <CALL STATEMENT> ;     */                          | 26814 SYNTHESIZ
|      ;                                                                        | 26814 SYNTHESIZ
| /*   <BASIC STATEMENT> ::= <GO TO STATEMENT> ;     */                         | 26822 SYNTHESIZ
|    DO;                                                                        | 26822 SYNTHESIZ
|      TBASIC = 6;                                                              | 26830 SYNTHESIZ
|    END;                                                                       | 26838 SYNTHESIZ
| /*   <BASIC STATEMENT> ::= <DECLARATION STATEMENT> ;     */                   | 26838 SYNTHESIZ
|      TBASIC=7;                                                                | 26838 SYNTHESIZ
| /*   <BASIC STATEMENT> ::= HALT ;     */                                      | 26854 SYNTHESIZ
|      DO;                                                                      | 26854 SYNTHESIZ
|        CALL QUADGEN(HALT);                                                    | 26862 SYNTHESIZ
|        TBASIC=8;                                                              | 26878 SYNTHESIZ
|      END;                                                                     | 26886 SYNTHESIZ
| /*   <BASIC STATEMENT> ::= ENABLE ;     */                                    | 26886 SYNTHESIZ
|      ;                                                                        | 26886 SYNTHESIZ
| /*   <BASIC STATEMENT> ::= DISABLE ;     */                                   | 26894 SYNTHESIZ
|      ;                                                                        | 26894 SYNTHESIZ
| /*   <BASIC STATEMENT> ::= ;     */                                           | 26902 SYNTHESIZ
|      ;                                                                        | 26902 SYNTHESIZ
| /*   <BASIC STATEMENT> ::= <LABEL DEFINITION> <BASIC STATEMENT>     */        | 26910 SYNTHESIZ
|      ;                                                                        | 26910 SYNTHESIZ
| /*   <IF STATEMENT> ::= <IF CLAUSE> <STATEMENT>     */                        | 26918 SYNTHESIZ
|     DO;                                                                       | 26918 SYNTHESIZ
|      IF TBASIC = 1 THEN DO;                                                   | 26926 SYNTHESIZ
|      CALL BRANCHLAB('.L'||SAVQUD(SAVQUDNO));                                  | 26942 SYNTHESIZ
|      CALL PUTQUAD(SAVQUD(SAVQUDNO),4,SAVEINDEX);                              | 26990 SYNTHESIZ
|      CALL PUTQUAD(SAVQUD(SAVQUDNO),1,BF);                                     | 27030 SYNTHESIZ
|      SAVQUDNO=SAVQUDNO-1;                                                     | 27070 SYNTHESIZ
|      CALL DEFINELAB(SP-1);                                                    | 27092 SYNTHESIZ
|      CALL QUADGEN(LAB);                                                       | 27102 SYNTHESIZ
|     END;                                                                      | 27118 SYNTHESIZ
|    IF TBASIC = 6 THEN DO;                                                     | 27118 SYNTHESIZ
|      I=GETQUAD(NEXTQUAD-1,4);                                                 | 27134 SYNTHESIZ
|      NEXTQUAD = NEXTQUAD-1;                                                   | 27166 SYNTHESIZ
|      CALL PUTQUAD(SAVQUD(SAVQUDNO),4,I);                                      | 27178 SYNTHESIZ
|      CALL PUTQUAD(SAVQUD(SAVQUDNO),1,BT);                                     | 27218 SYNTHESIZ
|      SAVQUDNO=SAVQUDNO-1;                                                     | 27258 SYNTHESIZ
|    END;                                                                       | 27270 SYNTHESIZ
|END;                                                                           | 27270 SYNTHESIZ
| /*   <IF STATEMENT> ::= <IF CLAUSE> <TRUE PART> <STATEMENT>     */            | 27270 SYNTHESIZ
|      IF SAVEQUAD2 ¬= 0 THEN DO;                                               | 27270 SYNTHESIZ
|        CALL BRANCHLAB('.L'||SAVEQUAD2);                                       | 27294 SYNTHESIZ
|        CALL PUTQUAD(SAVEQUAD2,4,SAVEINDEX);                                   | 27334 SYNTHESIZ
|        CALL DEFINELAB(SP-1);                                                  | 27366 SYNTHESIZ
|        CALL QUADGEN(LAB);                                                     | 27386 SYNTHESIZ
|      END;                                                                     | 27402 SYNTHESIZ
| /*   <IF STATEMENT> ::= <LABEL DEFINITION> <IF STATEMENT>     */              | 27402 SYNTHESIZ
|      ;                                                                        | 27402 SYNTHESIZ
| /*   <IF CLAUSE> ::= IF <EXPRESSION> THEN     */                              | 27410 SYNTHESIZ
|      DO;                                                                      | 27410 SYNTHESIZ
|      SAVQUDNO=SAVQUDNO+1;                                                     | 27418 SYNTHESIZ
|      SAVQUD(SAVQUDNO)=NEXTQUAD;                                                    | 27430 SYNTHESIZ
|        CALL QUADGEN(ZQ);                                                           | 27446 SYNTHESIZ
|END;                                                                                | 27462 SYNTHESIZ
| /*  <TRUE PART> ::= <BASIC STATEMENT> ELSE    */                                   | 27462 SYNTHESIZ
| DO;                                                                                | 27462 SYNTHESIZ
|    IF TBASIC = 1 THEN DO;                                                          | 27470 SYNTHESIZ
|     SAVEQUAD2=NEXTQUAD;                                                            | 27486 SYNTHESIZ
|     CALL QUADGEN(ZQ);                                                              | 27494 SYNTHESIZ
|     CALL PUTQUAD(NEXTQUAD-1,1,BR);                                                 | 27510 SYNTHESIZ
|     CALL PUTQUAD(NEXTQUAD-1,2,0);                                                  | 27546 SYNTHESIZ
|     CALL BRANCHLAB('.L'||SAVQUD(SAVQUDNO));                                        | 27580 SYNTHESIZ
|     CALL PUTQUAD(SAVQUD(SAVQUDNO),4,SAVEINDEX);                                    | 27628 SYNTHESIZ
|     CALL PUTQUAD(SAVQUD(SAVQUDNO),1,BF);                                           | 27668 SYNTHESIZ
|     SAVQUDNO=SAVQUDNO-1;                                                           | 27708 SYNTHESIZ
|     CALL DEFINELAB(SP-1);                                                          | 27720 SYNTHESIZ
|     CALL QUADGEN(LAB);                                                             | 27740 SYNTHESIZ
|    END;                                                                            | 27756 SYNTHESIZ
|    IF TBASIC = 6 THEN DO ;                                                         | 27756 SYNTHESIZ
|     SAVEQUAD2 = 0;                                                                 | 27772 SYNTHESIZ
|     I=GETQUAD(NEXTQUAD-1,4);                                                       | 27778 SYNTHESIZ
|     CALL PUTQUAD(SAVQUD(SAVQUDNO),4,I);                                            | 27810 SYNTHESIZ
|     CALL PUTQUAD(SAVQUD(SAVQUDNO),1,BT);                                           | 27850 SYNTHESIZ
|     SAVQUDNO=SAVQUDNO-1;                                                           | 27890 SYNTHESIZ
|     NEXTQUAD=NEXTQUAD-1;                                                           | 27902 SYNTHESIZ
|  END;                                                                              | 27914 SYNTHESIZ
|END;                                                                                | 27914 SYNTHESIZ
| /*  <GROUP> ::= <GROUP HEAD> <ENDING>     */                                       | 27914 SYNTHESIZ
|  DO;                                                                               | 27914 SYNTHESIZ
|     SP=SP+2;                                                                       | 27922 SYNTHESIZ
|     TABLE_LOC(SP)=LOOPINC;                                                         | 27934 SYNTHESIZ
|     TABLE_LOC(SP-2)=LOOP_INDEX;                                                    | 27950 SYNTHESIZ
|     CALL QUADGEN(ADD);                                                             | 27970 SYNTHESIZ
|     TABLE_LOC(SP)=TABLE_LOC(SP-2);                                                 | 27986 SYNTHESIZ
|     TABLE_LOC(SP-2)=LOOP_INDEX;                                                    | 28014 SYNTHESIZ
|     CALL QUADGEN(ASGN);                                                            | 28034 SYNTHESIZ
|     SP=SP-2;                                                                       | 28050 SYNTHESIZ
|     VAR(SP)=SAVLAB(SAVLABNO);                                                      | 28062 SYNTHESIZ
|     SAVLABNO=SAVLABNO-1;                                                           | 28086 SYNTHESIZ
|     CALL FINDLAB;                                                                  | 28098 SYNTHESIZ
|     CALL QUADGEN(ZQ);                                                              | 28106 SYNTHESIZ
|     CALL PUTQUAD(NEXTQUAD-1,1,BR);                                                 | 28122 SYNTHESIZ
|     CALL PUTQUAD(NEXTQUAD-1,2,0);                                                  | 28158 SYNTHESIZ
|     CALL PUTQUAD(NEXTQUAD-1,4,TABLE_LOC(SP));                                      | 28192 SYNTHESIZ
|     CALL BRANCHLAB('.L'||SAVQUD(SAVQUDNO));                                        | 28236 SYNTHESIZ
|     CALL PUTQUAD(SAVQUD(SAVQUDNO),4,TABLE_LOC(SP-1));                              | 28284 SYNTHESIZ
|     SAVQUDNO=SAVQUDNO-1;                                                           | 28336 SYNTHESIZ
|     CALL DEFINELAB(SP-1);                                                          | 28348 SYNTHESIZ
|     CALL QUADGEN(LAB);                                                             | 28368 SYNTHESIZ
|  END;                                                                              | 28384 SYNTHESIZ
| /*  <GROUP HEAD> ::= DO ;     */                                                   | 28384 SYNTHESIZ
|     ;                                                                              | 28384 SYNTHESIZ
| /*  <GROUP HEAD> ::= DO <STEP DEFINITION> ;     */                                 | 28392 SYNTHESIZ
|     ;                                                                              | 28392 SYNTHESIZ
| /*  <GROUP HEAD> ::= DO <WHILE CLAUSE> ;     */                                    | 28400 SYNTHESIZ
|     ;                                                                              | 28400 SYNTHESIZ
| /*  <GROUP HEAD> ::= DO <CASE SELECTOR> ;     */                                   | 28408 SYNTHESIZ
|     ;                                                                              | 28408 SYNTHESIZ
| /*  <GROUP HEAD> ::= <GROUP HEAD> <STATEMENT>     */                               | 28416 SYNTHESIZ
|     ;                                                                              | 28416 SYNTHESIZ
| /*  <STEP DEFINITION> ::= <VARIABLE> <REPLACE> <EXPRESSION> <ITERATION CONTROL>| 28424 SYNTHESIZ
|         */                                                                         | 28424 SYNTHESIZ
|     CALL GENLOOP;                                                                  | 28424 SYNTHESIZ
| /*  <ITERATION CONTROL> ::= <TO> <EXPRESSION>     */                               | 28440 SYNTHESIZ
|  DO;                                                                               | 28440 SYNTHESIZ
|     LOOPLIM=TABLE_LOC(SP);                                                         | 28448 SYNTHESIZ
|     FIXV(SP)=1;                                                                    | 28464 SYNTHESIZ
```

```
         CALL FINDNO;                                                                    | [illegible]
         LOOPINC=TABLE_LOC(SP);                                                          | 28488 SYNTHESIZE
   END;                                                                                  | 28504 SYNTHESIZE
/*  <ITERATION CONTROL> ::= <TO> <EXPRESSION> <BY> <EXPRESSION>     */                  | 28504 SYNTHESIZE
   DO;                                                                                   | 28504 SYNTHESIZE
         LOOPLIM=TABLE_LOC(SP-2);                                                        | 28512 SYNTHESIZE
         LOOPINC=TABLE_LOC(SP);                                                          | 28532 SYNTHESIZE
   END;                                                                                  | 28548 SYNTHESIZE
/*  <WHILE CLAUSE> ::= <WHILE> <EXPRESSION>     */                                       | 28548 SYNTHESIZE
    ;                                                                                    | 28549 SYNTHESIZE
/*  <CASE SELECTOR> ::= CASE <EXPRESSION>     */                                         | 28556 SYNTHESIZE
    ;                                                                                    | 28556 SYNTHESIZE
/*  <PROCEDURE DEFINITION> ::= <PROCEDURE HEAD> <STATEMENT LIST> <ENDING>     */         | 28564 SYNTHESIZ
    ;                                                                                    | 28564 SYNTHESIZ
/*  <PROCEDURE HEAD> ::= <PROCEDURE NAME> ;     */                                       | 28572 SYNTHESIZ
    ;                                                                                    | 28572 SYNTHESIZ
/*  <PROCEDURE HEAD> ::= <PROCEDURE NAME> <TYPE> ;     */                                | 28580 SYNTHESIZ
    ;                                                                                    | 28580 SYNTHESIZ
/*  <PROCEDURE HEAD> ::= <PROCEDURE NAME> <PARAMETER LIST> ;     */                      | 28588 SYNTHESIZ
    ;                                                                                    | 28588 SYNTHESIZ
/*  <PROCEDURE HEAD> ::= <PROCEDURE NAME> <PARAMETER LIST> <TYPE> ;     */               | 28596 SYNTHESIZ
    ;                                                                                    | 28596 SYNTHESIZ
/*  <PROCEDURE HEAD> ::= <PROCEDURE NAME> INTERRUPT <NUMBER> ;     */                    | 28604 SYNTHESIZ
    ;                                                                                    | 28604 SYNTHESIZ
/*  <PROCEDURE NAME> ::= <LABEL DEFINITION> PROCEDURE     */                             | 28612 SYNTHESIZ
    ;                                                                                    | 28612 SYNTHESIZ
/*  <PARAMETER LIST> ::= <PARAMETER HEAD> <IDENTIFIER> )     */                          | 28620 SYNTHESIZ
    ;                                                                                    | 28620 SYNTHESIZ
/*  <PARAMETER HEAD> ::= (     */                                                        | 28628 SYNTHESIZ
    ;                                                                                    | 28628 SYNTHESIZ
/*  <PARAMETER HEAD> ::= <PARAMETER HEAD> <IDENTIFIER> ,     */                          | 28636 SYNTHESIZ
    ;                                                                                    | 28636 SYNTHESIZ
/*  <ENDING> ::= END     */                                                              | 28644 SYNTHESIZ
    ;                                                                                    | 28644 SYNTHESIZ
/*  <ENDING> ::= END <IDENTIFIER>     */                                                 | 28652 SYNTHESIZ
    ;                                                                                    | 28652 SYNTHESIZ
/*  <ENDING> ::= <LABEL DEFINITION> <ENDING>     */                                      | 28660 SYNTHESIZ
    ;                                                                                    | 28660 SYNTHESIZ
/*  <LABEL DEFINITION> ::= <IDENTIFIER> :     */                                         | 28668 SYNTHESIZ
    DO;                                                                                  | 28668 SYNTHESIZ
         CALL DEFINELAB(SP-1);                                                           | 28676 SYNTHESIZ
          CALL QUADGEN(LAB);                                                             | 28696 SYNTHESIZ
    END;                                                                                 | 28712 SYNTHESIZ
/*  <LABEL DEFINITION> ::= <NUMBER> :     */                                             | 28712 SYNTHESIZ
    ;                                                                                    | 28712 SYNTHESIZ
/*  <RETURN STATEMENT> ::= RETURN     */                                                 | 28720 SYNTHESIZ
    ;                                                                                    | 28720 SYNTHESIZ
/*  <RETURN STATEMENT> ::= RETURN <EXPRESSION>     */                                    | 28728 SYNTHESIZ
    ;                                                                                    | 28728 SYNTHESIZ
/*  <CALL STATEMENT> ::= CALL <VARIABLE>     */                                          | 28736 SYNTHESIZ
    ;                                                                                    | 28736 SYNTHESIZ
/*  <GO TO STATEMENT> ::= <GO TO> <IDENTIFIER>     */                                    | 28744 SYNTHESIZ
    DO;                                                                                  | 28744 SYNTHESIZ
         CALL FINDLAB;                                                                   | 28752 SYNTHESIZ
         CALL QUADGEN(BR);                                                               | 28760 SYNTHESIZ
    END;                                                                                 | 28776 SYNTHESIZ
/*  <GO TO STATEMENT> ::= <GO TO> <NUMBER>     */                                        | 28776 SYNTHESIZ
    ;                                                                                    | 28776 SYNTHESIZ
/*  <GO TO> ::= GO TO     */                                                             | 28784 SYNTHESIZ
  DO;                                                                                    | 28784 SYNTHESIZ
    TBASIC = 6;                                                                          | 28792 SYNTHESIZ
  END;                                                                                   | 28800 SYNTHESIZ
/*  <GO TO> ::= GOTO     */                                                              | 28800 SYNTHESIZ
    ;                                                                                    | 28800 SYNTHESIZ
/*  <DECLARATION STATEMENT> ::= DECLARE <DECLARATION ELEMENT>     */                     | 28808 SYNTHESIZ
    ;                                                                                    | 28808 SYNTHESI
```

```
/*   <DECLARATION STATEMENT> ::= <DECLARATION STATEMENT> , <DECLARATION ELEMENT>| 28816 SYNTHESIZ
         */                                                                    | 28816 SYNTHESIZ
     ;                                                                         | 28816 SYNTHESIZ
/*   <DECLARATION ELEMENT> ::= <TYPE DECLARATION>    */                        | 28824 SYNTHESIZ
     ;                                                                         | 28824 SYNTHESIZ
/*   <DECLARATION ELEMENT> ::= <IDENTIFIER> LITERALLY <STRING>    */           | 28832 SYNTHESIZ
     ;                                                                         | 28832 SYNTHESIZ
/*   <DECLARATION ELEMENT> ::= <IDENTIFIER> <DATA LIST>    */                  | 28840 SYNTHESIZ
     ;                                                                         | 28840 SYNTHESIZ
/*   <DATA LIST> ::= <DATA HEAD> <CONSTANT> )    */                            | 28848 SYNTHESIZ
     ;                                                                         | 28848 SYNTHESIZ
/*   <DATA HEAD> ::= DATA (    */                                              | 28856 SYNTHESIZ
     ;                                                                         | 28856 SYNTHESIZ
/*   <DATA HEAD> ::= <DATA HEAD> <CONSTANT> ,    */                            | 28864 SYNTHESIZ
     ;                                                                         | 28864 SYNTHESIZ
/*   <TYPE DECLARATION> ::= <IDENTIFIER SPECIFICATION> <TYPE>    */            | 28872 SYNTHESIZ
     ;                                                                         | 28872 SYNTHESIZ
/*   <TYPE DECLARATION> ::= <BOUND HEAD> <NUMBER> ) <TYPE>    */               | 28880 SYNTHESIZ
     CALL SETDIM;                                                              | 28880 SYNTHESIZ
/*   <TYPE DECLARATION> ::= <TYPE DECLARATION> <INITIAL LIST>    */            | 28896 SYNTHESIZ
     ;                                                                         | 28896 SYNTHESIZ
/*   <TYPE> ::= BYTE    */                                                     | 28904 SYNTHESIZ
     ;                                                                         | 28904 SYNTHESIZ
/*   <TYPE> ::= ADDRESS    */                                                  | 28912 SYNTHESIZ
     ;                                                                         | 28912 SYNTHESIZ
/*   <TYPE> ::= LABEL    */                                                    | 28920 SYNTHESIZ
     ;                                                                         | 28920 SYNTHESIZ
/*   <BOUND HEAD> ::= <IDENTIFIER SPECIFICATION> (    */                       | 28928 SYNTHESIZ
     ;                                                                         | 28928 SYNTHESIZ
/*   <IDENTIFIER SPECIFICATION> ::= <VARIABLE NAME>    */                      | 28936 SYNTHESIZ
         NVARDEF=1;                                                            | 28936 SYNTHESIZ
/*   <IDENTIFIER SPECIFICATION> ::= <IDENTIFIER LIST> <VARIABLE NAME> )    */  | 28952 SYNTHESIZ
     ;                                                                         | 28952 SYNTHESIZ
/*   <IDENTIFIER LIST> ::= (    */                                             | 28960 SYNTHESIZ
         NVARDEF=1;                                                            | 28960 SYNTHESIZ
/*   <IDENTIFIER LIST> ::= <IDENTIFIER LIST> <VARIABLE NAME> ,    */           | 28976 SYNTHESIZ
         NVARDEF=NVARDEF+1;                                                    | 28976 SYNTHESIZ
/*   <VARIABLE NAME> ::= <IDENTIFIER>    */                                    | 28996 SYNTHESIZ
         CALL PUTIT(SP);                                                       | 28996 SYNTHESIZ
/*   <VARIABLE NAME> ::= <BASED VARIABLE> <IDENTIFIER>    */                   | 29020 SYNTHESIZ
     ;                                                                         | 29020 SYNTHESIZ
/*   <BASED VARIABLE> ::= <IDENTIFIER> BASED    */                             | 29028 SYNTHESIZ
     ;                                                                         | 29028 SYNTHESIZ
/*   <INITIAL LIST> ::= <INITIAL HEAD> <CONSTANT> )    */                      | 29036 SYNTHESIZ
 ;                                                                             | 29036 SYNTHESIZ
/*   <INITIAL HEAD> ::= INITIAL (    */                                        | 29044 SYNTHESIZ
     ;                                                                         | 29044 SYNTHESIZ
/*   <INITIAL HEAD> ::= <INITIAL HEAD> <CONSTANT> ,    */                      | 29052 SYNTHESIZ
     ;                                                                         | 29052 SYNTHESIZ
/*   <ASSIGNMENT> ::= <VARIABLE> <REPLACE> <EXPRESSION>    */                  | 29060 SYNTHESIZ
     DO;                                                                       | 29060 SYNTHESIZ
         CALL QUADGEN(ASGN);                                                   | 29066 SYNTHESIZ
     DEFITABLE_LOC(SP-2))=CARD_COUNT;                                          | 29084 SYNTHESIZ
     END;                                                                      | 29112 SYNTHESIZ
/*   <ASSIGNMENT> ::= <LEFT PART> <ASSIGNMENT>    */                           | 29112 SYNTHESIZ
     ;                                                                         | 29112 SYNTHESIZ
/*   <REPLACE> ::= =    */                                                     | 29120 SYNTHESIZ
     DO;                                                          /*NEWQUAD*/  | 29120 SYNTHESIZ
        SUBSFLAG=GETQUAD(NEXTQUAD-1,1);                           /*NEWQUAD*/  | 29128 SYNTHESIZ
        IF SUBSFLAG=SUBS THEN                                     /*NEWQUAD*/  | 29160 SYNTHESIZ
           CALL PUTQUAD(NEXTQUAD-1,1,SUBL);                       /*NEWQUAD*/  | 29184 SYNTHESIZ
        END;                                                      /*NEWQUAD*/  | 29212 SYNTHESIZ
/*   <LEFT PART> ::= <VARIABLE> ,    */                                        | 29212 SYNTHESIZ
     ;                                                                         | 29212 SYNTHESIZ
/*   <EXPRESSION> ::= <LOGICAL EXPRESSION>    */                               | 29220 SYNTHESIZ
     ;                                                                         | 29220 SYNTHESIZ
```

```
/*  <EXPRESSION> ::= <VARIABLE> := <LOGICAL EXPRESSION>    */                   | 29220 SYNTHESIZ
    ;                                                                           | 29228 SYNTHESIZ
/*  <LOGICAL EXPRESSION> ::= <LOGICAL FACTOR>    */                             | 29236 SYNTHESIZ
    ;                                                                           | 29236 SYNTHESIZ
/*  <LOGICAL EXPRESSION> ::= <LOGICAL EXPRESSION> OR <LOGICAL FACTOR>    */     | 29244 SYNTHESIZ
     CALL QUADGEN(OR);                                                          | 29244 SYNTHESIZ
/*  <LOGICAL EXPRESSION> ::= <LOGICAL EXPRESSION> XOR <LOGICAL FACTOR>    */    | 29268 SYNTHESIZ
    ;                                                                           | 29268 SYNTHESIZ
/*  <LOGICAL FACTOR> ::= <LOGICAL SECONDARY>    */                              | 29276 SYNTHESIZ
    ;                                                                           | 29276 SYNTHESIZ
/*  <LOGICAL FACTOR> ::= <LOGICAL FACTOR> AND <LOGICAL SECONDARY>    */         | 29284 SYNTHESIZ
     CALL QUADGEN(AND);                                                         | 29284 SYNTHESIZ
/*  <LOGICAL SECONDARY> ::= <LOGICAL PRIMARY>    */                             | 29308 SYNTHESIZ
    ;                                                                           | 29308 SYNTHESIZ
/*  <LOGICAL SECONDARY> ::= NOT <LOGICAL PRIMARY>    */                         | 29316 SYNTHESIZ
    ;                                                                           | 29316 SYNTHESIZ
/*  <LOGICAL PRIMARY> ::= <ARITHMETIC EXPRESSION>    */                         | 29324 SYNTHESIZ
    ;                                                                           | 29324 SYNTHESIZ
/*  <LOGICAL PRIMARY> ::= <ARITHMETIC EXPRESSION> <RELATION>                    | 29332 SYNTHESIZ
     <ARITHMETIC EXPRESSION>    */                                              | 29332 SYNTHESIZ
  DO;                                                                           | 29332 SYNTHESIZ
     CALL QUADGEN(REL);                                                         | 29340 SYNTHESIZ
     TBASIC = 1;                                                                | 29356 SYNTHESIZ
  END;                                                                          | 29364 SYNTHESIZ
/*  <RELATION> ::= =    */                                                      | 29364 SYNTHESIZ
         FIXV(SP)=EQ;                                                           | 29364 SYNTHESIZ
/*  <RELATION> ::= <    */                                                      | 29388 SYNTHESIZ
         FIXV(SP)=LT;                                                           | 29388 SYNTHESIZ
/*  <RELATION> ::= >    */                                                      | 29412 SYNTHESIZ
         FIXV(SP)=GT;                                                           | 29412 SYNTHESIZ
/*  <RELATION> ::= <COMP>    */                                                 | 29436 SYNTHESIZ
    ;                                                                           | 29436 SYNTHESIZ
/*  <COMP> ::= < >    */                                                        | 29444 SYNTHESIZ
         FIXV(SP-1)=NE;                                                         | 29444 SYNTHESIZ
/*  <COMP> ::= < =    */                                                        | 29472 SYNTHESIZ
         FIXV(SP-1)=LE;                                                         | 29472 SYNTHESIZ
/*  <COMP> ::= > =    */                                                        | 29500 SYNTHESIZ
         FIXV(SP-1)=GE;                                                         | 29500 SYNTHESIZ
/*  <ARITHMETIC EXPRESSION> ::= <TERM>    */                                    | 29528 SYNTHESIZ
    ;                                                                           | 29528 SYNTHESIZ
/*  <ARITHMETIC EXPRESSION> ::= <ARITHMETIC EXPRESSION> + <TERM>    */          | 29536 SYNTHESIZ
         CALL QUADGEN(ADD);                                                     | 29536 SYNTHESIZ
/*  <ARITHMETIC EXPRESSION> ::= <ARITHMETIC EXPRESSION> - <TERM>    */          | 29560 SYNTHESIZ
         CALL QUADGEN(SUB);                                                     | 29560 SYNTHESIZ
/*  <ARITHMETIC EXPRESSION> ::= <ARITHMETIC EXPRESSION> PLUS <TERM>    */       | 29584 SYNTHESIZ
         CALL QUADGEN(ADD);                                                     | 29584 SYNTHESIZ
/*  <ARITHMETIC EXPRESSION> ::= <ARITHMETIC EXPRESSION> MINUS <TERM>    */      | 29608 SYNTHESIZ
         CALL QUADGEN(SUB);                                                     | 29608 SYNTHESIZ
/*  <ARITHMETIC EXPRESSION> ::= - <TERM>    */                                  | 29632 SYNTHESIZ
    DO;                                                                         | 29632 SYNTHESIZ
     CALL QUADGEN(UMIN);                                                        | 29640 SYNTHESIZ
         CALL MOVESTACK (SP,SP-1);                                              | 29656 SYNTHESIZ
    END;                                                                        | 29684 SYNTHESIZ
/*  <TERM> ::= <PRIMARY>    */                                                  | 29684 SYNTHESIZ
    ;                                                                           | 29684 SYNTHESIZ
/*  <TERM> ::= <TERM> * <PRIMARY>    */                                         | 29692 SYNTHESIZ
         CALL QUADGEN(MUL);                                                     | 29692 SYNTHESIZ
/*  <TERM> ::= <TERM> / <PRIMARY>    */                                         | 29716 SYNTHESIZ
         CALL QUADGEN(DIV);                                                     | 29716 SYNTHESIZ
/*  <TERM> ::= <TERM> MOD <PRIMARY>    */                                       | 29740 SYNTHESIZ
         CALL QUADGEN(MMOD);                                                    | 29740 SYNTHESIZ
/*  <PRIMARY> ::= <CONSTANT>    */                                              | 29764 SYNTHESIZ
         CALL FINDNO;                                                           | 29764 SYNTHESIZ
/*  <PRIMARY> ::= . <CONSTANT>    */                                            | 29780 SYNTHESIZ
    ;                                                                           | 29780 SYNTHESIZ
/*  <PRIMARY> ::= <CONSTANT HEAD> <CONSTANT> )    */                            | 29788 SYNTHESIZ
```

```
|     ;                                                               | 29788 SYNTHESIZE
| /*  <PRIMARY> ::= <VARIABLE>     */                                 | 29796 SYNTHESIZE
|     ;                                                               | 29796 SYNTHESIZE
| /*  <PRIMARY> ::= . <VARIABLE>     */                               | 29804 SYNTHESIZE
|     ;                                                               | 29804 SYNTHESIZE
| /*  <PRIMARY> ::= ( <EXPRESSION> )     */                           | 29812 SYNTHESIZE
|           CALL MOVESTACK (SP-1,SP-2);                               | 29812 SYNTHESIZE
| /*  <CONSTANT HEAD> ::= . (     */                                  | 29852 SYNTHESIZE
|     ;                                                               | 29852 SYNTHESIZE
| /*  <CONSTANT HEAD> ::= <CONSTANT HEAD> <CONSTANT> ,     */         | 29860 SYNTHESIZE
|     ;                                                               | 29860 SYNTHESIZE
| /*  <VARIABLE> ::= <IDENTIFIER>     */                              | 29868 SYNTHESIZE
|           CALL FINDIT(SP);                                          | 29868 SYNTHESIZE
| /*  <VARIABLE> ::= <SUBSCRIPT HEAD> <EXPRESSION> )     */           | 29892 SYNTHESIZE
|     CALL QUADGEN(SUBS);                                             | 29892 SYNTHESIZE
| /*  <SUBSCRIPT HEAD> ::= <IDENTIFIER> (     */                      | 29916 SYNTHESIZE
|     DO;                                                             | 29916 SYNTHESIZE
|       CALL FINDIT(SP-1);                                            | 29924 SYNTHESIZE
|       I=TABLE_LOC(SP-1);                                            | 29944 SYNTHESIZE
|          IF SIZE(I) = 1 THEN                                        | 29964 SYNTHESIZE
|            DO;                                                      | 29996 SYNTHESIZE
|                OUTPUT=' VARIABLE '||SYMB(I)||' NOT A DIMENSIONED ARRAY';   | 29988 SYNTHESIZE
|          END;                                                       | 30050 SYNTHESIZE
|     END;                                                            | 30050 SYNTHESIZE
| /*  <SUBSCRIPT HEAD> ::= <SUBSCRIPT HEAD> <EXPRESSION> ,     */      | 30050 SYNTHESIZE
|     ;                                                               | 30050 SYNTHESIZE
| /*  <CONSTANT> ::= <STRING>     */                                  | 30058 SYNTHESIZE
|     ;                                                               | 30058 SYNTHESIZE
| /*  <CONSTANT> ::= <NUMBER>     */                                  | 30066 SYNTHESIZE
|     ;                                                               | 30066 SYNTHESIZE
| /*  <TO> ::= TO     */                                              | 30074 SYNTHESIZE
| /*  <BY> ::= BY     */                                              | 30074 SYNTHESIZE
|     ;                                                               | 30074 SYNTHESIZE
| /*  <WHILE> ::= WHILE     */                                        | 30082 SYNTHESIZE
|     ;                                                               | 30082 SYNTHESIZE
|   END;                                                              | 30090 SYNTHESIZE
|END SYNTHESIZE;                                                      | 30098 SYNTHESIZE
|                                                                     | 30104
|                                                                     | 30104
|                                                                     | 30104
|                                                                     | 30104
|  /*                 SYNTACTIC PARSING FUNCTIONS                  */ | 30104
|                                                                     | 30104
|                                                                     | 30104
|RIGHT_CONFLICT:                                                      | 30104
|   PROCEDURE (LEFT) BIT(1);                                          | 30104
|      DECLARE LEFT FIXED;                                            | 30104 RIGHT_CONFLIC
|      /*  THIS PROCEDURE IS TRUE IF TOKEN IS   A LEGAL RIGHT CONTEXT OF LEFT*/ | 30116 RIGHT_CONFLIC
|      RETURN ("C0" & SHL(BYTE(C1(LEFT), SHR(TOKEN,2)), SHL(TOKEN,1)) | 30116 RIGHT_CONFLIC
|         & "06")) = 0;                                               | 30144 RIGHT_CONFLIC
|   END RIGHT_CONFLICT;                                               | 30186 RIGHT_CONFLIC
|                                                                     | 30192
|                                                                     | 30192
|RECOVER:                                                             | 30192
|   PROCEDURE;                                                        | 30192
|      /* IF THIS IS THE SECOND SUCCESSIVE CALL TO RECOVER, DISCARD ONE SYMBOL */| 30192 RECOVER
|      IF ¬ FAILSOFT THEN CALL SCAN;                                  | 30192 RECOVER
|      FAILSOFT = FALSE;                                              | 30230 RECOVER
|      DO WHILE ¬ STOPIT(TOKEN);                                      | 30236 RECOVER
|         CALL SCAN;  /* TO FIND SOMETHING SOLID IN THE TEXT  */      | 30262 RECOVER
|      END;                                                           | 30266 RECOVER
|      DO WHILE RIGHT_CONFLICT (PARSE_STACK(SP));                     | 30274 RECOVER
|         IF SP > 2 THEN SP = SP - 1;  /* AND IN THE STACK  */        | 30308 RECOVER
|         ELSE CALL SCAN;  /* BUT DON'T GO TOO FAR  */                | 30336 RECOVER
|      END;                                                           | 30348 RECOVER
|      OUTPUT = 'RESUME:' || SUBSTR(POINTER, TEXT_LIMIT-CP+MARGIN_CHOP+7); | 30356 RECOVER
```

```
   END RECOVER;                                                                [illegible] RECOVER

STACKING:                                                                      30434
   PROCEDURE BIT(1);  /* STACKING DECISION FUNCTION */                         30434
      CALLCOUNT(1) = CALLCOUNT(1) + 1;                                         30434 STACKING
      DO FOREVER;     /* UNTIL RETURN  */                                      30474 STACKIN
         DO CASE SHR(BYTE(C1(PARSE_STACK(SP)),SHR(TOKEN,2)),SHL(3-TOKEN,1)&6)&3;  30474 STACKIN

            /*  CASE 0  */                                                     30546 STACKIN
            DO;     /* ILLEGAL SYMBOL PAIR  */                                 30546 STACKIN
               CALL ERROR('ILLEGAL SYMBOL PAIR: ' || V(PARSE_STACK(SP)) || X1||  30546 STACKIN
                  V(TOKEN), 1);                                                30596 STACKIN
               CALL STACK_DUMP;                                                30636 STACKIN
               CALL RECOVER;                                                   30644 STACKIN
            END;                                                               30652 STACKIN
                                                                               30652 STACKIN
            /*  CASE 1  */                                                     30652 STACKIN
                                                                               30652 STACKIN
            RETURN TRUE;       /*  STACK TOKEN  */                             30652 STACKIN
                                                                               30670 STACKIN
            /*  CASE 2  */                                                     30670 STACKIN
                                                                               30670 STACKIN
            RETURN FALSE;      /* DON'T STACK IT YET  */                       30670 STACKIN
                                                                               30686 STACKIN
            /*  CASE 3  */                                                     30686 STACKIN
                                                                               30686 STACKIN
            DO;       /* MUST CHECK TRIPLES  */                                30686 STACKIN
               J = SHL(PARSE_STACK(SP-1), 16) + SHL(PARSE_STACK(SP), 8) + TOKEN;  30694 STACKIN
               I = -1;  K = NC1TRIPLES + 1;  /* BINARY SEARCH OF TRIPLES  */   30736 STACKIN
               DO WHILE I + 1 < K;                                             30756 STACKIN
                  L = SHR(I+K, 1);                                             30776 STACKIN
                  IF C1TRIPLES(L) > J THEN K = L;                              30792 STACKIN
                  ELSE IF C1TRIPLES(L) < J THEN I = L;                         30824 STACKIN
                  ELSE RETURN TRUE;  /* IT IS A VALID TRIPLE  */               30864 STACKIN
               END;                                                            30882 STACKIN
               RETURN FALSE;                                                   30890 STACKIN
            END;                                                               30898 STACKIN
                                                                               30898 STACKIN
         END;     /* OF DO CASE  */                                            30898 STACKIN
      END;  /*  OF DO FOREVER */                                               30906 STACKIN
   END STACKING;                                                               30914 STACKIN
                                                                               30920
PR_OK:                                                                         30920
   PROCEDURE(PRD) BIT(1);                                                      30920
      /* DECISION PROCEDURE FOR CONTEXT CHECK OF EQUAL OR IMBEDDED RIGHT PARTS*/  30920 PR_OK
      DECLARE (H, I, J, PRD) FIXED;                                            30920 PR_OK
      DO CASE CONTEXT_CASE(PRD);                                               30932 PR_OK
                                                                               30962 PR_OK
         /*  CASE 0 -- NO CHECK REQUIRED  */                                   30962 PR_OK
                                                                               30962 PR_OK
         RETURN TRUE;                                                          30962 PR_OK
                                                                               30972 PR_OK
         /*  CASE 1 -- RIGHT CONTEXT CHECK  */                                 30972 PR_OK
                                                                               30972 PR_OK
         RETURN ¬ RIGHT_CONFLICT (HDTB(PRD));                                  30972 PR_OK
                                                                               31012 PR_OK
         /*  CASE 2 -- LEFT CONTEXT CHECK  */                                  31012 PR_OK
                                                                               31012 PR_OK
         DO;                                                                   31012 PR_OK
            H = HDTB(PRD) - NT;                                                31020 PR_OK
            I = PARSE_STACK(SP - PRLENGTH(PRD));                               31038 PR_OK
            DO J = LEFT_INDEX(H-1) TO LEFT_INDEX(H) - 1;                       31064 PR_OK
               IF LEFT_CONTEXT(J) = I THEN RETURN TRUE;                        31128 PR_OK
            END;                                                               31160 PR_OK
            RETURN FALSE;                                                      31168 PR_OK
         END;                                                                  31176 PR_OK
```

```
|                                                                          | 31176 PR_OK
|        /*  CASE 3 -- CHECK TRIPLES  */                                   | 31176 PR_OK
|                                                                          | 31176 PR_OK
|        DO;                                                               | 31176 PR_OK
|           H = HDTB(PRD) - NT;                                            | 31184 PR_OK
|           I = SHL(PARSE_STACK(SP - PRLENGTH(PRD)), 8) + TOKEN;           | 31202 PR_OK
|           DO J = TRIPLE_INDEX(H-1) TO TRIPLE_INDEX(H) - 1;               | 31236 PR_OK
|              IF CONTEXT_TRIPLE(J) = I THEN RETURN TRUE;                  | 31300 PR_OK
|           END;                                                           | 31334 PR_OK
|           RETURN FALSE;                                                  | 31342 PR_OK
|        END;                                                              | 31350 PR_OK
|                                                                          | 31350 PR_OK
|     END;  /* OF DO CASE  */                                              | 31350 PR_OK
|  END PR_OK;                                                              | 31358 PR_OK
|                                                                          | 31364
|                                                                          | 31364
|  /*                          ANALYSIS ALGORITHM                       */ | 31364
|                                                                          | 31364
|                                                                          | 31364
|                                                                          | 31364
|REDUCE:                                                                   | 31364
|  PROCEDURE;                                                              | 31364
|     DECLARE (I, J, PRD) FIXED;                                           | 31364 REDUCE
|     /* PACK STACK TOP INTO ONE WORD */                                   | 31376 REDUCE
|     DO I = SP - 4 TO SP - 1;                                             | 31376 REDUCE
|        J = SHL(J, 8) + PARSE_STACK(I);                                   | 31420 REDUCE
|     END;                                                                 | 31452 REDUCE
|                                                                          | 31460 REDUCE
|     DO PRD = PR_INDEX(PARSE_STACK(SP)-1) TO PR_INDEX(PARSE_STACK(SP)) - 1; | 31460 REDUCE
|        IF (PPMASK(PRLENGTH(PRD)) & J) = PRTB(PRD) THEN                   | 31536 REDUCE
|           IF PR_OK(PRD) THEN                                             | 31586 REDUCE
|           DO;  /* AN ALLOWED REDUCTION */                                | 31586 REDUCE
|              MP = SP - PRLENGTH(PRD) + 1; MPP1 = MP + 1;                 | 31606 REDUCE
|              CALL SYNTHESIZE(PRDTB(PRD));                                | 31642 REDUCE
|              SP = MP;                                                    | 31664 REDUCE
|              PARSE_STACK(SP) = HDTB(PRD);                                | 31672 REDUCE
|              RETURN;                                                     | 31690 REDUCE
|           END;                                                           | 31696 REDUCE
|     END;                                                                 | 31696 REDUCE
|                                                                          | 31704 REDUCE
|     /* LOOK UP HAS FAILED, ERROR CONDITION */                            | 31704 REDUCE
|     CALL ERROR('NO PRODUCTION IS APPLICABLE',1);                         | 31704 REDUCE
|     CALL STACK_DUMP;                                                     | 31724 REDUCE
|     FAILSOFT = FALSE;                                                    | 31732 REDUCE
|     CALL RECOVER;                                                        | 31738 REDUCE
|  END REDUCE;                                                             | 31746 REDUCE
|                                                                          | 31752
|COMPILATION_LOOP:                                                         | 31752
|  PROCEDURE;                                                              | 31752
|                                                                          | 31752 COMPILATION_LO
|     COMPILING = TRUE;                                                    | 31752 COMPILATION_LO
|     DO WHILE COMPILING;     /* ONCE AROUND FOR EACH PRODUCTION (REDUCTION)  */| 31772 COMPILATION_LO
|        DO WHILE STACKING;                                                | 31784 COMPILATION_LO
|           SP = SP + 1;                                                   | 31804 COMPILATION_LO
|           IF SP = STACKSIZE THEN                                         | 31816 COMPILATION_LO
|              DO;                                                         | 31840 COMPILATION_LO
|                 CALL ERROR ('STACK OVERFLOW *** CHECKING ABORTED ***', 2); | 31832 COMPILATION_LO
|                 RETURN;   /* THUS ABORTING CHECKING */                   | 31852 COMPILATION_LO
|              END;                                                        | 31858 COMPILATION_LO
|           PARSE_STACK(SP) = TOKEN;                                       | 31858 COMPILATION_LO
|           VAR(SP) = BCD;                                                 | 31870 COMPILATION_LO
|           FIXV(SP) = NUMBER_VALUE;                                       | 31886 COMPILATION_LO
|           CALL SCAN;                                                     | 31902 COMPILATION_LO
|        END;                                                              | 31906 COMPILATION_LO
|                                                                          | 31914 COMPILATION_LO
|        CALL REDUCE;                                                      | 31914 COMPILATION_LO
```

```
              /* OF [illegible] WHILE COMPILING */                                      | 31922 COMPILATION_LOO
   END COMPILATION_LOOP;                                                                | 31930 COMPILATION_LOO
                                                                                        | 31936
                                                                                        | 31936
                                                                                        | 31936
                                                                                        | 31936
PRINT_SUMMARY:                                                                          | 31936
   PROCEDURE;                                                                           | 31936
      DECLARE I FIXED;                                                                  | 31936 PRINT_SUMMARY
      CALL PRINT_DATE_AND_TIME ('END OF CHECKING ', DATE, TIME);                        | 31948 PRINT_SUMMARY
      OUTPUT = '';                                                                      | 31993 PRINT_SUMMARY
      OUTPUT = CARD_COUNT || ' CARDS WERE CHECKED.';                                    | 32016 PRINT_SUMMARY
      IF ERROR_COUNT = 0 THEN OUTPUT = 'NO ERRORS WERE DETECTED.';                      | 32062 PRINT_SUMMARY
      ELSE IF ERROR_COUNT > 1 THEN                                                      | 32098 PRINT_SUMMARY
         OUTPUT = ERROR_COUNT || ' ERRORS (' || SEVERE_ERRORS ||                        | 32130 PRINT_SUMMARY
               ' SEVERE) WERE DETECTED.';                                               | 32174 PRINT_SUMMARY
      ELSE IF SEVERE_ERRORS = 1 THEN OUTPUT = 'ONE SEVERE ERROR WAS DETECTED.';         | 32208 PRINT_SUMMARY
         ELSE OUTPUT = 'ONE ERROR WAS DETECTED.';                                       | 32252 PRINT_SUMMARY
      IF PREVIOUS_ERROR > 0 THEN                                                        | 32280 PRINT_SUMMARY
         OUTPUT = 'THE LAST DETECTED ERROR WAS ON LINE ' || PREVIOUS_ERROR ||           | 32304 PRINT_SUMMARY
               PERIOD;                                                                  | 32324 PRINT_SUMMARY
      IF CONTROL(BYTE('D')) THEN CALL DUMPIT;                                           | 32358 PRINT_SUMMARY
      DOUBLE_SPACE;                                                                     | 32384 PRINT_SUMMARY
      CLOCK(3) = TIME;                                                                  | 32408 PRINT_SUMMARY
      DO I = 1 TO 3;    /* WATCH OUT FOR MIDNIGHT */                                    | 32434 PRINT_SUMMARY
         IF CLOCK(I) < CLOCK(I-1) THEN CLOCK(I) = CLOCK(I) +  8640000;                  | 32470 PRINT_SUMMARY
      END;                                                                              | 32534 PRINT_SUMMARY
      CALL PRINT_TIME ('TOTAL TIME IN CHECKER     ', CLOCK(3) - CLOCK(0));              | 32542 PRINT_SUMMARY
      CALL PRINT_TIME ('SET UP TIME               ', CLOCK(1) - CLOCK(0));              | 32580 PRINT_SUMMARY
      CALL PRINT_TIME ('ACTUAL CHECKING TIME      ', CLOCK(2) - CLOCK(1));              | 32618 PRINT_SUMMARY
      CALL PRINT_TIME ('CLEAN_UP TIME AT END      ', CLOCK(3) - CLOCK(2));              | 32658 PRINT_SUMMARY
      IF CLOCK(2) > CLOCK(1) THEN    /* WATCH OUT FOR CLOCK BEING OFF */                | 32698 PRINT_SUMMARY
      OUTPUT = 'CHECKING RATE: ' || 6000*CARD_COUNT/(CLOCK(2)-CLOCK(1)) ||              | 32738 PRINT_SUMMARY
            ' CARDS PER MINUTE.';                                                       | 32794 PRINT_SUMMARY
   END PRINT_SUMMARY;                                                                   | 32828 PRINT_SUMMARY
                                                                                        | 32834
MAIN_PROCEDURE:                                                                         | 32834
   PROCEDURE;                                                                           | 32834
      CLOCK(0) = TIME;  /* KEEP TRACK OF TIME IN EXECUTION */                           | 32834 MAIN_PROCEDURE
      CALL INITIALIZATION;                                                              | 32870 MAIN_PROCEDURE
                                                                                        | 32874 MAIN_PROCEDURE
      CLOCK(1) = TIME;                                                                  | 32874 MAIN_PROCEDURE
                                                                                        | 32900 MAIN_PROCEDURE
      CALL COMPILATION_LOOP;                                                            | 32900 MAIN_PROCEDURE
                                                                                        | 32908 MAIN_PROCEDURE
      CLOCK(2) = TIME;                                                                  | 32908 MAIN_PROCEDURE
                                                                                        | 32934 MAIN_PROCEDURE
      /* CLOCK(3) GETS SET IN PRINT_SUMMARY */                                          | 32934 MAIN_PROCEDURE
      CALL PRINT_SUMMARY;                                                               | 32934 MAIN_PROCEDURE
         FILE(1,RCD_BUFF)=QUADS;                                                        | 32942 MAIN_PROCEDURE
         QUADS= FILE(1,0);                                                              | 32964 MAIN_PROCEDURE
         RCD_BUFF=0;                                                                    | 32992 MAIN_PROCEDURE
      CALL PRINTSYMB;                                                                   | 32998 MAIN_PROCEDURE
         CALL PROCESS_QUADS;                                                 /*NEWQUAD*/ | 33006 MAIN_PROCEDURE
     CALL PRINT_NEWQUADS;                                                    /*NEWQUAD*/ | 33014 MAIN_PROCEDURE
                                                                                        | 33022 MAIN_PROCEDURE
   END MAIN_PROCEDURE;                                                                  | 33022 MAIN_PROCEDURE
                                                                                        | 33028
                                                                                        | 33028
CALL MAIN_PROCEDURE;                                                                    | 33028
RETURN SEVERE_ERRORS;                                                                   | 33036
                                                                                        | 33046
EOF EOF EOF                                                                             | 33046
```

```
*  FILE CONTROL BLOCK  39000     26000     3     2     13000     7056     12788
*  LOAD FILE WRITTEN.
END OF COMPILATION APRIL 4, 1979.  CLOCK TIME = 3:7:15.79.

3355 CARDS CONTAINING 1702 STATEMENTS WERE COMPILED.
NO ERRORS WERE DETECTED.
33054 BYTES OF PROGRAM, 18173 OF DATA. 2736 OF DESCRIPTORS. 4878 OF STRINGS.
 TOTAL CORE REQUIREMENT 58841 BYTES.

SYMBOL  TABLE  DUMP


ADD                  : FIXED     AT 3156(10),   DECLARED ON LINE 475 AND REFERENCED 3 TIMES.
ADDR SS              : BIT(8)    AT 345(9),     DECLARED ON LINE 534 AND REFERENCED 4 TIMES.
ALLOC_REG_OP         : LABEL     AT 17788(14),  DECLARED ON LINE 1963 AND REFERENCED 6 TIMES.
  PARAMETER  1       : CHARACTER AT 2492(13),   DECLARED ON LINE 1971 AND REFERENCED 7 TIMES.
ALLOC_REG_RES        : LABEL     AT 17154(14),  DECLARED ON LINE 1904 AND REFERENCED 2 TIMES.
  PARAMETER  1       : CHARACTER AT 2484(13),   DECLARED ON LINE 1908 AND REFERENCED 3 TIMES.
ALPHA                : CHARACTER AT 1996(13),   DECLARED ON LINE 552 AND REFERENCED 2 TIMES.
ALPHABET             : CHARACTER AT 912(13),    DECLARED ON LINE 376 AND REFERENCED 2 TIMES.
AND                  : FIXED     AT 3232(10),   DECLARED ON LINE 494 AND REFERENCED 1 TIMES.
ASGN                 : FIXED     AT 3220(10),   DECLARED ON LINE 491 AND REFERENCED 4 TIMES.
B_R1                 : LABEL     AT 8112(14),   DECLARED ON LINE 1135 AND REFERENCED 1 TIMES.
B_R2                 : LABEL     AT 7270(14),   DECLARED ON LINE 1068 AND REFERENCED 2 TIMES.
BA_RO_READ           : LABEL     AT 6930(14),   DECLARED ON LINE 1043 AND REFERENCED 3 TIMES.
BCD                  : CHARACTER AT 896(13),    DECLARED ON LINE 351 AND REFERENCED 6 TIMES.
BF                   : FIXED     AT 3188(10),   DECLARED ON LINE 483 AND REFERENCED 3 TIMES.
BR                   : FIXED     AT 3130(10),   DECLARED ON LINE 481 AND REFERENCED 4 TIMES.
BR_FLAG              : BIT(8)    AT 12(9),      DECLARED ON LINE 520 AND REFERENCED 10 TIMES.
BRANCHLAB            : LABEL     AT 22642(14),  DECLARED ON LINE 2398 AND REFERENCED 5 TIMES.
  PARAMETER  1       : CHARACTER AT 2560(13),   DECLARED ON LINE 2400 AND REFERENCED 2 TIMES.
BT                   : FIXED     AT 3184(10),   DECLARED ON LINE 482 AND REFERENCED 3 TIMES.
BUF_MIC              : CHARACTER AT 1808(13),   DECLARED ON LINE 523 AND REFERENCED 5 TIMES.
BUFFER               : CHARACTER AT 916(13),    DECLARED ON LINE 386 AND REFERENCED 73 TIMES.
BZ                   : FIXED     AT 3228(10),   DECLARED ON LINE 493 AND REFERENCED 1 TIMES.
CALLCOUNT            : FIXED     AT 424(10),    DECLARED ON LINE 421 AND REFERENCED 6 TIMES.
CARD_COUNT           : FIXED     AT 260(10),    DECLARED ON LINE 387 AND REFERENCED 8 TIMES.
CHANGE               : BIT(8)    AT 123(9),     DECLARED ON LINE 528 AND REFERENCED 8 TIMES.
CHAR                 : LABEL     AT 2332(14),   DECLARED ON LINE 672 AND REFERENCED 2 TIMES.
CHARTYPE             : BIT(8)    AT 3564(11),   DECLARED ON LINE 371 AND REFERENCED 6 TIMES.
CLOCK                : FIXED     AT 508(10),    DECLARED ON LINE 425 AND REFERENCED 20 TIMES.
COMPILATION_LOOP     : LABEL     AT 31760(14),  DECLARED ON LINE 3274 AND REFERENCED 1 TIMES.
COMPILING            : BIT(8)    AT 393(10),    DECLARED ON LINE 407 AND REFERENCED 4 TIMES.
CONLOC               : FIXED     AT 3296(10),   DECLARED ON LINE 512 AND REFERENCED 1 TIMES.
CONTEXT_CASE         : BIT(8)    AT 3182(11),   DECLARED ON LINE 321 AND REFERENCED 1 TIMES.
CONTEXT_TRIPLE       : FIXED     AT 3376(11),   DECLARED ON LINE 331 AND REFERENCED 1 TIMES.
CONTROL              : BIT(8)    AT 4076(11),   DECLARED ON LINE 372 AND REFERENCED 8 TIMES.
CONVAL               : FIXED     AT 3500(10),   DECLARED ON LINE 512 AND REFERENCED 16 TIMES.
CONVAL_INDEX         : LABEL     AT 11672(14),  DECLARED ON LINE 1478 AND REFERENCED 1 TIMES.
CORE                 : FIXED     AT 3260(10),   DECLARED ON LINE 501 AND REFERENCED 0 TIMES.
CP                   : FIXED     AT 3552(11),   DECLARED ON LINE 351 AND REFERENCED 33 TIMES.
CURR_MIC             : FIXED     AT 16(9),      DECLARED ON LINE 521 AND REFERENCED 63 TIMES.
CI                   : CHARACTER AT 460(13),    DECLARED ON LINE 250 AND REFERENCED 2 TIMES.
CITRIPLES            : FIXED     AT 1356(11),   DECLARED ON LINE 261 AND REFERENCED 2 TIMES.
DEALLOC_TEMP         : LABEL     AT 16110(14),  DECLARED ON LINE 1822 AND REFERENCED 5 TIMES.
  PARAMETER  1       : CHARACTER AT 2476(13),   DECLARED ON LINE 1828 AND REFERENCED 2 TIMES.
  PARAMETER  2       : FIXED     AT 1156(9),    DECLARED ON LINE 1827 AND REFERENCED 12 TIMES.
DEALLOCABL           : BIT(8)    AT 392(9),     DECLARED ON LINE 537 AND REFERENCED 9 TIMES.
DEALLOCATE           : LABEL     AT 15032(14),  DECLARED ON LINE 1725 AND REFERENCED 2 TIMES.
DEF                  : FIXED     AT 1880(10),   DECLARED ON LINE 466 AND REFERENCED 4 TIMES.
DEFINELAB            : LABEL     AT 24790(14),  DECLARED ON LINE 2569 AND REFERENCED 6 TIMES.
  PARAMETER  1       : FIXED     AT 1580(8),    DECLARED ON LINE 2574 AND REFERENCED 4 TIMES.
DIV                  : FIXED     AT 3168(10),   DECLARED ON LINE 478 AND REFERENCED 1 TIMES.
DIVIDE               : FIXED     AT 288(10),    DECLARED ON LINE 396 AND REFERENCED 2 TIMES.
```

```
DIVLOC               : FIXED     AT 3292(10),     DECLARED ON LINE 510 AND REFERENCED 0 TIMES.
DO_SWITCH            : FIXED     AT 3284(10),     DECLARED ON LINE 509 AND REFERENCED 0 TIMES.
DOUBLE               : CHARACTER AT 904(13),      DECLARED ON LINE 355 AND REFERENCED 9 TIMES.
DUMPIT               : LABEL     AT 5250(14),     DECLARED ON LINE 910 AND REFERENCED 1 TIMES.
EOFILE               : FIXED     AT 292(10),      DECLARED ON LINE 396 AND REFERENCED 4 TIMES.
EQ                   : FIXED     AT 3196(10),     DECLARED ON LINE 485 AND REFERENCED 1 TIMES.
ERROR                : LABEL     AT 1526(14),     DECLARED ON LINE 611 AND REFERENCED 6 TIMES.
  PARAMETER  1       : CHARACTER AT 2128(13),     DECLARED ON LINE 615 AND REFERENCED 1 TIMES.
  PARAMETER  2       : FIXED     AT 56(8),        DECLARED ON LINE 615 AND REFERENCED 1 TIMES.
ERROR_COUNT          : FIXED     AT 264(10),      DECLARED ON LINE 387 AND REFERENCED 5 TIMES.
FAILSOFT             : BIT(8)    AT 397(10),      DECLARED ON LINE 407 AND REFERENCED 4 TIMES.
FIND_OP              : LABEL     AT 17488(14),    DECLARED ON LINE 1935 AND REFERENCED 1 TIMES.
  PARAMETER  1       : CHARACTER AT 2488(13),     DECLARED ON LINE 1943 AND REFERENCED 1 TIMES.
FIND_RES             : LABEL     AT 16540(14),    DECLARED ON LINE 1855 AND REFERENCED 2 TIMES.
  PARAMETER  1       : CHARACTER AT 2480(13),     DECLARED ON LINE 1864 AND REFERENCED 2 TIMES.
FINDIT               : LABEL     AT 25626(14),    DECLARED ON LINE 2644 AND REFERENCED 2 TIMES.
  PARAMETER  1       : FIXED     AT 1624(8),      DECLARED ON LINE 2646 AND REFERENCED 3 TIMES.
FINDLAB              : LABEL     AT 22344(14),    DECLARED ON LINE 2370 AND REFERENCED 2 TIMES.
FINDNO               : LABEL     AT 25834(14),    DECLARED ON LINE 2664 AND REFERENCED 2 TIMES.
FIXV                 : FIXED     AT 624(10),      DECLARED ON LINE 445 AND REFERENCED 14 TIMES.
GE                   : FIXED     AT 3216(10),     DECLARED ON LINE 490 AND REFERENCED 1 TIMES.
GENLOOP              : LABEL     AT 26208(14),    DECLARED ON LINE 2701 AND REFERENCED 1 TIMES.
GET_CARD             : LABEL     AT 1876(14),     DECLARED ON LINE 642 AND REFERENCED 4 TIMES.
GETQUAD              : LABEL     AT 21780(14),    DECLARED ON LINE 2314 AND REFERENCED 3 TIMES.
  PARAMETER  1       : FIXED     AT 1496(8),      DECLARED ON LINE 2316 AND REFERENCED 2 TIMES.
  PARAMETER  2       : FIXED     AT 1500(8),      DECLARED ON LINE 2316 AND REFERENCED 1 TIMES.
GT                   : FIXED     AT 3204(10),     DECLARED ON LINE 487 AND REFERENCED 2 TIMES.
HALT                 : FIXED     AT 3176(10),     DECLARED ON LINE 480 AND REFERENCED 4 TIMES.
HOTB                 : BIT(8)    AT 2922(11),     DECLARED ON LINE 307 AND REFERENCED 4 TIMES.
I                    : FIXED     AT 532(10),      DECLARED ON LINE 433 AND REFERENCED 131 TIMES.
I_FORMAT             : CHARACTER PROCEDURE AT 1400(14),  DECLARED ON LINE 501 AND REFERENCED 2 TIMES.
  PARAMETER  1       : FIXED     AT 40(8),        DECLARED ON LINE 603 AND REFERENCED 1 TIMES.
  PARAMETER  2       : FIXED     AT 44(8),        DECLARED ON LINE 603 AND REFERENCED 2 TIMES.
IDENT                : FIXED     AT 280(10),      DECLARED ON LINE 395 AND REFERENCED 3 TIMES.
II                   : FIXED     AT 500(9),       DECLARED ON LINE 542 AND REFERENCED 96 TIMES.
INIT                 : FIXED     AT 2688(10),     DECLARED ON LINE 466 AND REFERENCED 3 TIMES.
INITIALIZATION       : LABEL     AT 4084(14),     DECLARED ON LINE 648 AND REFERENCED 1 TIMES.
J                    : FIXED     AT 536(10),      DECLARED ON LINE 433 AND REFERENCED 17 TIMES.
JJ                   : FIXED     AT 504(9),       DECLARED ON LINE 542 AND REFERENCED 31 TIMES.
K                    : FIXED     AT 540(10),      DECLARED ON LINE 433 AND REFERENCED 6 TIMES.
KK                   : FIXED     AT 508(9),       DECLARED ON LINE 542 AND REFERENCED 7 TIMES.
L                    : FIXED     AT 544(10),      DECLARED ON LINE 433 AND REFERENCED 10 TIMES.
LAB                  : FIXED     AT 3248(10),     DECLARED ON LINE 495 AND REFERENCED 9 TIMES.
LAB_INDEX            : LABEL     AT 11566(14),    DECLARED ON LINE 1467 AND REFERENCED 3 TIMES.
LABDEF               : FIXED     AT 1052(10),     DECLARED ON LINE 456 AND REFERENCED 5 TIMES.
LABID                : CHARACTER AT 1248(13),     DECLARED ON LINE 457 AND REFERENCED 12 TIMES.
LASTREF              : FIXED     AT 948(10),      DECLARED ON LINE 458 AND REFERENCED 1 TIMES.
LAVS                 : FIXED     AT 124(9),       DECLARED ON LINE 526 AND REFERENCED 23 TIMES.
LE                   : FIXED     AT 3212(10),     DECLARED ON LINE 489 AND REFERENCED 1 TIMES.
LEFT_CONTEXT         : BIT(8)    AT 3212(11),     DECLARED ON LINE 327 AND REFERENCED 1 TIMES.
LEFT_INDEX           : BIT(8)    AT 3317(11),     DECLARED ON LINE 328 AND REFERENCED 2 TIMES.
LL                   : FIXED     AT 512(9),       DECLARED ON LINE 542 AND REFERENCED 72 TIMES.
LOC_QUAD             : FIXED     AT 8(9),         DECLARED ON LINE 519 AND REFERENCED 15 TIMES.
LOCAT                : FIXED     AT 1476(10),     DECLARED ON LINE 466 AND REFERENCED 4 TIMES.
LOOP_INDEX           : FIXED     AT 3276(10),     DECLARED ON LINE 505 AND REFERENCED 4 TIMES.
LOOPINC              : FIXED     AT 3272(10),     DECLARED ON LINE 504 AND REFERENCED 3 TIMES.
LOOPLIM              : FIXED     AT 3268(10),     DECLARED ON LINE 503 AND REFERENCED 3 TIMES.
LT                   : FIXED     AT 3200(10),     DECLARED ON LINE 486 AND REFERENCED 2 TIMES.
MAIN_PROCEDURE       : LABEL     AT 32842(14),    DECLARED ON LINE 3329 AND REFERENCED 1 TIMES.
MARGIN_CHOP          : FIXED     AT 3560(11),     DECLARED ON LINE 361 AND REFERENCED 6 TIMES.
MAXREG               : FIXED     AT 496(9),       DECLARED ON LINE 541 AND REFERENCED 8 TIMES.
MIC_GEN              : LABEL     AT 8992(14),     DECLARED ON LINE 1208 AND REFERENCED 1 TIMES.
MIC_LOC              : FIXED     AT 20(9),        DECLARED ON LINE 522 AND REFERENCED 3 TIMES.
MMOD                 : FIXED     AT 3172(10),     DECLARED ON LINE 479 AND REFERENCED 2 TIMES.
MOVESTACK            : LABEL     AT 26100(14),    DECLARED ON LINE 2688 AND REFERENCED 2 TIMES.
  PARAMETER  1       : FIXED     AT 1648(8),      DECLARED ON LINE 2690 AND REFERENCED 4 TIMES.
```

```
   PARAMETER  2          : FIXED     AT 1652(8),     DECLARED ON LINE 2590 AND REFERENCED 4 TIMES.
MP                       : FIXED     AT 932(10),     DECLARED ON LINE 451 AND REFERENCED 4 TIMES.
MPP1                     : FIXED     AT 936(10),     DECLARED ON LINE 451 AND REFERENCED 1 TIMES.
MUL                      : FIXED     AT 3160(10),    DECLARED ON LINE 476 AND REFERENCED 2 TIMES.
MULLOC                   : FIXED     AT 3288(10),    DECLARED ON LINE 510 AND REFERENCED 0 TIMES.
NCONSTANT                : FIXED     AT 3152(10),    DECLARED ON LINE 474 AND REFERENCED 6 TIMES.
NE                       : FIXED     AT 3208(10),    DECLARED ON LINE 488 AND REFERENCED 1 TIMES.
NEWQUAD                  : FIXED     AT 608(9),      DECLARED ON LINE 547 AND REFERENCED 39 TIMES.
NEWQUAD_GEN              : LABEL     AT 18662(14),   DECLARED ON LINE 2039 AND REFERENCED 1 TIMES.
NEWQUADNO                : FIXED     AT 0(8),        DECLARED ON LINE 548 AND REFERENCED 7 TIMES.
NEXT_VAR                 : FIXED     AT 172(9),      DECLARED ON LINE 529 AND REFERENCED 25 TIMES.
NEXTQUAD                 : FIXED     AT 1168(10),    DECLARED ON LINE 462 AND REFERENCED 28 TIMES.
NLABEL                   : FIXED     AT 944(10),     DECLARED ON LINE 455 AND REFERENCED 22 TIMES.
NN                       : FIXED     AT 516(9),      DECLARED ON LINE 542 AND REFERENCED 0 TIMES.
NO_OP                    : LABEL     AT 8482(14),    DECLARED ON LINE 1164 AND REFERENCED 2 TIMES.
NOT_LETTER_OR_DIGIT: BIT(8)          AT 0(10),       DECLARED ON LINE 372 AND REFERENCED 4 TIMES.
NSYMBOL                  : FIXED     AT 3148(10),    DECLARED ON LINE 473 AND REFERENCED 22 TIMES.
NUMBER                   : FIXED     AT 284(10),     DECLARED ON LINE 396 AND REFERENCED 2 TIMES.
NUMBER_VALUE             : FIXED     AT 276(10),     DECLARED ON LINE 392 AND REFERENCED 4 TIMES.
NVARDEF                  : FIXED     AT 3144(10),    DECLARED ON LINE 472 AND REFERENCED 5 TIMES.
OPER                     : FIXED     AT 20(8),       DECLARED ON LINE 551 AND REFERENCED 22 TIMES.
OPERAND1                 : CHARACTER AT 1816(13),    DECLARED ON LINE 525 AND REFERENCED 23 TIMES.
OPERAND2                 : CHARACTER AT 1820(13),    DECLARED ON LINE 525 AND REFERENCED 17 TIMES.
OPERATION                : CHARACTER AT 1812(13),    DECLARED ON LINE 525 AND REFERENCED 2 TIMES.
OPRND1                   : FIXED     AT 8(8),        DECLARED ON LINE 550 AND REFERENCED 16 TIMES.
OPRND2                   : FIXED     AT 12(8),       DECLARED ON LINE 550 AND REFERENCED 14 TIMES.
OPRTOR                   : FIXED     AT 4(8),        DECLARED ON LINE 550 AND REFERENCED 14 TIMES.
OR                       : FIXED     AT 3236(10),    DECLARED ON LINE 495 AND REFERENCED 2 TIMES.
OUT_MIC                  : LABEL     AT 6182(14),    DECLARED ON LINE 980 AND REFERENCED 21 TIMES.
PAD                      : CHARACTER PROCEDURE AT 1290(14),    DECLARED ON LINE 592 AND REFERENCED 28 TIMES.
   PARAMETER  1          : CHARACTER AT 2120(13),    DECLARED ON LINE 594 AND REFERENCED 3 TIMES.
   PARAMETER  2          : FIXED     AT 28(8),       DECLARED ON LINE 594 AND REFERENCED 2 TIMES.
PAGE                     : CHARACTER AT 900(13),     DECLARED ON LINE 355 AND REFERENCED 1 TIMES.
PARSE_STACK              : BIT(8)    AT 548(10),     DECLARED ON LINE 442 AND REFERENCED 16 TIMES.
PERIOD                   : CHARACTER AT 940(13),     DECLARED ON LINE 430 AND REFERENCED 1 TIMES.
POINT                    : FIXED     AT 464(9),      DECLARED ON LINE 540 AND REFERENCED 17 TIMES.
POINTER                  : CHARACTER AT 928(13),     DECLARED ON LINE 419 AND REFERENCED 2 TIMES.
PR_INDEX                 : BIT(8)    AT 3438(11),    DECLARED ON LINE 335 AND REFERENCED 2 TIMES.
PR_OK                    : LABEL     AT 30928(14),   DECLARED ON LINE 3203 AND REFERENCED 1 TIMES.
   PARAMETER  1          : FIXED     AT 2244(8),     DECLARED ON LINE 3206 AND REFERENCED 6 TIMES.
PROTB                    : BIT(8)    AT 2792(11),    DECLARED ON LINE 299 AND REFERENCED 1 TIMES.
PREVIOUS_ERROR           : FIXED     AT 272(10),     DECLARED ON LINE 337 AND REFERENCED 4 TIMES.
PRINT_DATE_AND_TIME: LABEL           AT 3788(14),    DECLARED ON LINE 827 AND REFERENCED 3 TIMES.
   PARAMETER  1          : CHARACTER AT 2212(13),    DECLARED ON LINE 829 AND REFERENCED 1 TIMES.
   PARAMETER  2          : FIXED     AT 212(8),      DECLARED ON LINE 829 AND REFERENCED 2 TIMES.
   PARAMETER  3          : FIXED     AT 216(8),      DECLARED ON LINE 829 AND REFERENCED 1 TIMES.
PRINT_NEWQUADS           : LABEL     AT 12688(14),   DECLARED ON LINE 1547 AND REFERENCED 1 TIMES.
PRINT_SUMMARY            : LABEL     AT 31944(14),   DECLARED ON LINE 3299 AND REFERENCED 1 TIMES.
PRINT_TIME               : LABEL     AT 3479(14),    DECLARED ON LINE 817 AND REFERENCED 5 TIMES.
   PARAMETER  1          : CHARACTER AT 2188(13),    DECLARED ON LINE 819 AND REFERENCED 5 TIMES.
   PARAMETER  2          : FIXED     AT 188(8),      DECLARED ON LINE 819 AND REFERENCED 7 TIMES.
PRINTSYMB                : LABEL     AT 20506(14),   DECLARED ON LINE 2259 AND REFERENCED 1 TIMES.
PRLENGTH                 : BIT(8)    AT 3052(11),    DECLARED ON LINE 315 AND REFERENCED 4 TIMES.
PRMASK                   : FIXED     AT 400(10),     DECLARED ON LINE 413 AND REFERENCED 1 TIMES.
PROCESS_QUADS            : LABEL     AT 20294(14),   DECLARED ON LINE 2241 AND REFERENCED 1 TIMES.
PRTB                     : FIXED     AT 2272(11),    DECLARED ON LINE 290 AND REFERENCED 1 TIMES.
PUT_MIC                  : LABEL     AT 5848(14),    DECLARED ON LINE 958 AND REFERENCED 7 TIMES.
   PARAMETER  1          : FIXED     AT 500(8),      DECLARED ON LINE 964 AND REFERENCED 3 TIMES.
PUT_NEWQUAD              : LABEL     AT 11354(14),   DECLARED ON LINE 1446 AND REFERENCED 13 TIMES.
PUTIT                    : LABEL     AT 25248(14),   DECLARED ON LINE 2613 AND REFERENCED 1 TIMES.
   PARAMETER  1          : FIXED     AT 1608(8),     DECLARED ON LINE 2615 AND REFERENCED 4 TIMES.
PUTQUAD                  : LABEL     AT 21956(14),   DECLARED ON LINE 2330 AND REFERENCED 18 TIMES.
   PARAMETER  1          : FIXED     AT 1508(8),     DECLARED ON LINE 2332 AND REFERENCED 2 TIMES.
   PARAMETER  2          : FIXED     AT 1512(8),     DECLARED ON LINE 2332 AND REFERENCED 1 TIMES.
   PARAMETER  3          : FIXED     AT 1516(8),     DECLARED ON LINE 2332 AND REFERENCED 1 TIMES.
PUTTEMP                  : LABEL     AT 22134(14),   DECLARED ON LINE 2349 AND REFERENCED 5 TIMES.
```

```
[illegible]
QUADGEN            : LABEL     AT 22306(14),  DECLARED ON LINE 2415 AND REFERENCED 29 TIMES.
  PARAMETER  1     : FIXED     AT 1552(8),    DECLARED ON LINE 2421 AND REFERENCED 12 TIMES.
QUADS              : FIXED     AT 3704(10),   DECLARED ON LINE 516 AND REFERENCED 18 TIMES.
RCD_BUFF           : FIXED     AT 0(9),       DECLARED ON LINE 517 AND REFERENCED 18 TIMES.
RCD_NR             : FIXED     AT 4(9),       DECLARED ON LINE 518 AND REFERENCED 11 TIMES.
R0                 : BIT(8)    AT 336(9),     DECLARED ON LINE 532 AND REFERENCED 10 TIMES.
READQUAD           : LABEL     AT 11794(14),  DECLARED ON LINE 1489 AND REFERENCED 1 TIMES.
  PARAMETER  1     : FIXED     AT 836(8),     DECLARED ON LINE 1493 AND REFERENCED 4 TIMES.
RECOVER            : LABEL     AT 30200(14),  DECLARED ON LINE 3148 AND REFERENCED 2 TIMES.
REDUCE             : LABEL     AT 31372(14),  DECLARED ON LINE 3247 AND REFERENCED 1 TIMES.
REFNO              : FIXED     AT 520(9),     DECLARED ON LINE 543 AND REFERENCED 9 TIMES.
REFRENCE           : FIXED     AT 400(9),     DECLARED ON LINE 538 AND REFERENCED 5 TIMES.
REGNO              : FIXED     AT 580(9),     DECLARED ON LINE 546 AND REFERENCED 12 TIMES.
REL                : FIXED     AT 3192(10),   DECLARED ON LINE 484 AND REFERENCED 4 TIMES.
RESERVED_LIMIT     : FIXED     AT 3556(11),   DECLARED ON LINE 361 AND REFERENCED 3 TIMES.
RESULT             : CHARACTER AT 1824(13),   DECLARED ON LINE 525 AND REFERENCED 21 TIMES.
RIGHT_CONFLICT     : LABEL     AT 30112(14),  DECLARED ON LINE 3139 AND REFERENCED 2 TIMES.
  PARAMETER  1     : FIXED     AT 2192(8),    DECLARED ON LINE 3141 AND REFERENCED 1 TIMES.
RSLT               : FIXED     AT 16(8),      DECLARED ON LINE 550 AND REFERENCED 14 TIMES.
RTEMP_MINUS_1      : LABEL     AT 8704(14),   DECLARED ON LINE 1186 AND REFERENCED 1 TIMES.
RTEMP_SHIFT        : LABEL     AT 7928(14),   DECLARED ON LINE 1119 AND REFERENCED 2 TIMES.
RTEMP_UNIBUS       : LABEL     AT 6876(14),   DECLARED ON LINE 1031 AND REFERENCED 2 TIMES.
RTEMP_4            : LABEL     AT 7388(14),   DECLARED ON LINE 1078 AND REFERENCED 1 TIMES.
R0_PLUS2_READ      : LABEL     AT 6414(14),   DECLARED ON LINE 994 AND REFERENCED 3 TIMES.
R2_SHIFT           : LABEL     AT 8222(14),   DECLARED ON LINE 1144 AND REFERENCED 1 TIMES.
R3_D               : LABEL     AT 7120(14),   DECLARED ON LINE 1055 AND REFERENCED 3 TIMES.
R3_R3_PLUS_B       : LABEL     AT 8524(14),   DECLARED ON LINE 1171 AND REFERENCED 1 TIMES.
R3_UNIBUS          : LABEL     AT 6726(14),   DECLARED ON LINE 1019 AND REFERENCED 3 TIMES.
R3_0               : LABEL     AT 7716(14),   DECLARED ON LINE 1103 AND REFERENCED 1 TIMES.
S                  : CHARACTER AT 924(13),    DECLARED ON LINE 409 AND REFERENCED 12 TIMES.
SAVEINDEX          : FIXED     AT 3264(10),   DECLARED ON LINE 502 AND REFERENCED 4 TIMES.
SAVELAB            : CHARACTER AT 1800(13),   DECLARED ON LINE 507 AND REFERENCED 0 TIMES.
SAVELOC            : FIXED     AT 3280(10),   DECLARED ON LINE 506 AND REFERENCED 1 TIMES.
SAVEQUAD           : FIXED     AT 1160(10),   DECLARED ON LINE 460 AND REFERENCED 0 TIMES.
SAVEQUAD2          : FIXED     AT 1164(10),   DECLARED ON LINE 461 AND REFERENCED 5 TIMES.
SAVEREF            : FIXED     AT 940(10),    DECLARED ON LINE 453 AND REFERENCED 0 TIMES.
SAVEVAR            : CHARACTER AT 1804(13),   DECLARED ON LINE 508 AND REFERENCED 1 TIMES.
SAVLAB             : CHARACTER AT 1756(13),   DECLARED ON LINE 469 AND REFERENCED 2 TIMES.
SAVLABNO           : FIXED     AT 3140(10),   DECLARED ON LINE 470 AND REFERENCED 6 TIMES.
SAVQUD             : FIXED     AT 3092(10),   DECLARED ON LINE 467 AND REFERENCED 14 TIMES.
SAVQUDNO           : FIXED     AT 3136(10),   DECLARED ON LINE 468 AND REFERENCED 28 TIMES.
SCAN               : LABEL     AT 2380(14),   DECLARED ON LINE 681 AND REFERENCED 5 TIMES.
SET_BIT            : LABEL     AT 5664(14),   DECLARED ON LINE 937 AND REFERENCED 183 TIMES.
  PARAMETER  1     : FIXED     AT 392(8),     DECLARED ON LINE 941 AND REFERENCED 1 TIMES.
SET_FIELD          : LABEL     AT 5698(14),   DECLARED ON LINE 944 AND REFERENCED 22 TIMES.
  PARAMETER  1     : FIXED     AT 400(8),     DECLARED ON LINE 948 AND REFERENCED 3 TIMES.
  PARAMETER  2     : FIXED     AT 404(8),     DECLARED ON LINE 948 AND REFERENCED 1 TIMES.
  PARAMETER  3     : FIXED     AT 408(8),     DECLARED ON LINE 948 AND REFERENCED 2 TIMES.
SETDIM             : LABEL     AT 25122(14),  DECLARED ON LINE 2600 AND REFERENCED 1 TIMES.
SEVERE_ERRORS      : FIXED     AT 268(10),    DECLARED ON LINE 387 AND REFERENCED 6 TIMES.
SIZE               : FIXED     AT 2284(10),   DECLARED ON LINE 466 AND REFERENCED 5 TIMES.
SORT               : LABEL     AT 14174(14),  DECLARED ON LINE 1667 AND REFERENCED 2 TIMES.
  PARAMETER  1     : FIXED     AT 1040(8),    DECLARED ON LINE 1672 AND REFERENCED 2 TIMES.
  PARAMETER  2     : BIT(8)    AT 1036(8),    DECLARED ON LINE 1671 AND REFERENCED 2 TIMES.
SORTNUM            : FIXED     AT 552(9),     DECLARED ON LINE 545 AND REFERENCED 14 TIMES.
SORTREF            : FIXED     AT 524(9),     DECLARED ON LINE 544 AND REFERENCED 14 TIMES.
SP                 : FIXED     AT 928(10),    DECLARED ON LINE 451 AND REFERENCED 117 TIMES.
STACK_DUMP         : LABEL     AT 5450(14),   DECLARED ON LINE 921 AND REFERENCED 3 TIMES.
STACKING           : LABEL     AT 30442(14),  DECLARED ON LINE 3163 AND REFERENCED 1 TIMES.
STATUS             : BIT(8)    AT 337(9),     DECLARED ON LINE 533 AND REFERENCED 11 TIMES.
STOPIT             : BIT(8)    AT 296(10),    DECLARED ON LINE 407 AND REFERENCED 3 TIMES.
STOREQUAD          : LABEL     AT 21576(14),  DECLARED ON LINE 2295 AND REFERENCED 1 TIMES.
  PARAMETER  1     : FIXED     AT 1472(8),    DECLARED ON LINE 2297 AND REFERENCED 2 TIMES.
  PARAMETER  2     : FIXED     AT 1476(8),    DECLARED ON LINE 2297 AND REFERENCED 1 TIMES.
  PARAMETER  3     : FIXED     AT 1480(8),    DECLARED ON LINE 2297 AND REFERENCED 1 TIMES.
```

AD-A068 950 GEORGE WASHINGTON UNIV WASHINGTON D C SCHOOL OF ENGI--ETC F/G 9/2
DEVELOPMENT OF EXPERIMENTAL COMPILERS TO GENERATE EMULATORS FOR--ETC(U)
APR 79 R E MERWIN DASG60-78-C-0115
UNCLASSIFIED NL

2 OF 2
AD
A068950

END
DATE
FILMED
7-79
DDC

```
  PARAMETER  4      : FIXED     AT 1494(8),    DECLARED ON LINE 2297 AND REFERENCED 1 TIMES.
  PARAMETER  5      : FIXED     AT 1498(8),    DECLARED ON LINE 2297 AND REFERENCED 1 TIMES.
SUB                 : FIXED     AT 3164(10),   DECLARED ON LINE 477 AND REFERENCED 2 TIMES.
SUBL                : FIXED     AT 3252(10),   DECLARED ON LINE 499 AND REFERENCED 2 TIMES.
SUBS                : FIXED     AT 3224(10),   DECLARED ON LINE 492 AND REFERENCED 5 TIMES.
SUBSFLAG            : FIXED     AT 356(9),     DECLARED ON LINE 535 AND REFERENCED 4 TIMES.
SYMB                : CHARACTER AT 1352(13),   DECLARED ON LINE 465 AND REFERENCED 23 TIMES.
SYMB_INDEX          : LABEL     AT 11460(14),  DECLARED ON LINE 1456 AND REFERENCED 4 TIMES.
SYNTHESIZE          : LABEL     AT 26656(14),  DECLARED ON LINE 2732 AND REFERENCED 1 TIMES.
  PARAMETER  1      : FIXED     AT 1664(8),    DECLARED ON LINE 2734 AND REFERENCED 1 TIMES.
TABLE_LOC           : FIXED     AT 1172(10),   DECLARED ON LINE 463 AND REFERENCED 52 TIMES.
TBASIC              : FIXED     AT 1156(10),   DECLARED ON LINE 459 AND REFERENCED 10 TIMES.
TEMP                : FIXED     AT 432(9),     DECLARED ON LINE 539 AND REFERENCED 10 TIMES.
TEMP_CHAR           : CHARACTER AT 1992(13),   DECLARED ON LINE 549 AND REFERENCED 10 TIMES.
TEXT                : CHARACTER AT 920(13),    DECLARED ON LINE 336 AND REFERENCED 15 TIMES.
TEXT_LIMIT          : FIXED     AT 255(10),    DECLARED ON LINE 387 AND REFERENCED 12 TIMES.
TGOTO               : FIXED     AT 3256(10),   DECLARED ON LINE 500 AND REFERENCED 0 TIMES.
TOKEN               : FIXED     AT 3548(11),   DECLARED ON LINE 351 AND REFERENCED 14 TIMES.
TRIPLE_INDEX        : BIT(8)    AT 3390(11),   DECLARED ON LINE 332 AND REFERENCED 2 TIMES.
TX                  : BIT(8)    AT 3820(11),   DECLARED ON LINE 371 AND REFERENCED 3 TIMES.
UMIN                : FIXED     AT 3240(10),   DECLARED ON LINE 496 AND REFERENCED 2 TIMES.
V                   : CHARACTER AT 24(13),     DECLARED ON LINE 126 AND REFERENCED 9 TIMES.
V_INDEX             : BIT(8)    AT 1340(11),   DECLARED ON LINE 148 AND REFERENCED 4 TIMES.
VAR                 : CHARACTER AT 944(13),    DECLARED ON LINE 444 AND REFERENCED 20 TIMES.
VARIBLS             : CHARACTER AT 1828(13),   DECLARED ON LINE 527 AND REFERENCED 10 TIMES.
VARNUM              : FIXED     AT 360(9),     DECLARED ON LINE 536 AND REFERENCED 22 TIMES.
WRITE_REGS          : LABEL     AT 18376(14),  DECLARED ON LINE 2015 AND REFERENCED 4 TIMES.
X1                  : CHARACTER AT 932(13),    DECLARED ON LINE 429 AND REFERENCED 3 TIMES.
X4                  : CHARACTER AT 936(13),    DECLARED ON LINE 429 AND REFERENCED 1 TIMES.
X70                 : CHARACTER AT 908(13),    DECLARED ON LINE 357 AND REFERENCED 3 TIMES.
ZQ                  : FIXED     AT 3244(10),   DECLARED ON LINE 497 AND REFERENCED 5 TIMES.
```

```
MACRO DEFINITIONS:

NT                      LITERALLY: 51
NSY                     LITERALLY: 108
TRUE                    LITERALLY: 1
FALSE                   LITERALLY: 0
LABELS                  LITERALLY: 25
DX_SIZE                 LITERALLY: 500
FOREVER                 LITERALLY: WHILE 1
SYMBOLS                 LITERALLY: 100
MAXQUADS                LITERALLY: 220
STACKSIZE               LITERALLY: 75
CONSTANTS               LITERALLY: 50
DISKWORDS               LITERALLY: 900
NC1TRIPLES              LITERALLY: 228
EJECT_PAGE              LITERALLY: OUTPUT(1) = PAGE
DOUBLE_SPACE            LITERALLY: OUTPUT(1) = DOUBLE

IDCOMPARES          = 343105
SYMBOL TABLE SIZE = 311
MACRO DEFINITIONS = 15
STACKING DECISIONS= 57026
SCAN                = 16603
EMITRR              = 697
EMITRX              = 8520
FORCEACCUMULATOR    = 3475
ARITHEMIT           = 541
GENSTORE            = 809
FIXBFW              = 467
FIXDATAWORD         = 12
FIXCHW              = 844
GETDATA             = 6
GETCODE             = 4
FINDADDRESS         = 1033
SHORTCFIX           = 838
LONGCFIX            = 6
SHORTDFIX           = 9
LONGDFIX            = 3
FREE STRING AREA    = 92014

REGISTER VALUES (RELATIVE TO R11):
R4 = 0
R5 = 0
R6 = 0
R7 = 0
R8 = 15852
R9 = 11640
R10 = 4332
R11 = 0
R12 = 0
R13 = 18168

 INSTRUCTION FREQUENCIES:

BALR     92
BCTR     3
BCR      133
LPR      1
LTR      14
LCR      15
NR       4
```

```
OR      8
XR      16
LR      111
CR      3
AR      30
SR      257
DR      2
ALR     4
SLR     4
STH     3
LA      737
STC     76
IC      82
EX      18
BAL     636
BC      733
LH      6
ST      1605
N       25
O       1
X       5
L       2802
C       209
A       209
S       148
M       6
D       24
AL      2
SRL     31
SLL     449
SRA     16
SRDA    25
STM     88
TM      7
DI      1
LM      89

TOTAL TIME IN COMPILER    0:6:26.58.
SET UP TIME               0:0:5.83.
ACTUAL COMPILATION TIME   0:5:52.10.
POST-COMPILATION TIME     0:0:28.65.
COMPILATION RATE: 571 CARDS PER MINUTE.
```

APPENDIX 7.2 PLM REPRESENTATION OF TEST PROGRAMS

```
FIBS:PROCEDURE;
     DECLARE (FIB,FIB1,FIB2,I,N) BYTE;
     FIB1 = 0;
     FIB2 = 1;
     N = 100;
     DO I = 1 TO N;
          FIB = FIB1;
          FIB1 = FIB2;
          FIB2 = FIB + FIB1;
          END;
  END;
EOF EOF EOF
```

(a) FIBONACCI SERIES

```
GRT_ELM:PROCEDURE;
       DECLARE (I,N,GRTST_ELMNT,INDEX) BYTE;
       DECLARE ARRAY(10) BYTE;
       N=10;
       GRTST_ELMNT=ARRAY(1);
       INDEX=1;
       DO I=2 TO N;
             IF GRTST_ELMNT > ARRAY(I) THEN GO TO LAB1;
                   GRTST_ELMNT=ARRAY(I);
                   INDEX=I;
         LAB1:END;
       END;
EOF
```

(b) GREATEST ELEMENT

```
PRIME:PROCEDURE;
     DECLARE (I,J,K) BYTE;
     DECLARE A(256) BYTE;
     DO I=1 TO 256;
         A(I)=1;
         END;
     DO I=2 TO 256;
         IF A(I)=0 THEN GO TO LAB1;
             K=2*I;
             DO J=K TO 256 BY I;
                 A(J)=0;
                 END;
         LAB1:END;
     END;
EOF
```

(c) PRIME IDENTIFIER

```
SORT:PROCEDURE;
       DECLARE ARRAY(10) BYTE;
       DECLARE (I,J,K,TEMP,SWITCH) BYTE;
       SWITCH = 0;
       K=0;
  LAB2:K = K + 1;
       DO I=K TO 9;
            J=I+1;
            IF ARRAY(I) < ARRAY(J) THEN GO TO LAB1;
                 SWITCH=1;
                 TEMP=ARRAY(I);
                 ARRAY(I)=ARRAY(J);
                 ARRAY(J)=TEMP;
            LAB1:END;
            IF SWITCH = 1 THEN GO TO LAB2;
            END SORT;
EOF
```

(d) BUBBLE SORT

```
FILTER:PROCEDURE;
     DECLARE(AT,EXAT,I,J,N,C)BYTE;
     DECLARE A(20) BYTE;
     DECLARE B(20) BYTE;
     B(1)=0;
     DO I=1 TO N;
         J=I+1;
         A(I)=A(I)*AT;
         B(J)=B(I)*EXAT;
         C=A(I)+B(J);
         B(J)=C;
         END;
     END;
EOF
```

(e) DIGITAL FILTER

APPENDIX 7.3-1 QUAD AND R QUAD REPRESENTATION OF TEST PROGRAMS

```
            QUADS GENERATED

OPERATOR  OPERAND1/  OPERAND2 RESULT/   ****  RCD_NR  LOC_QUAD
          CONDITION           LABEL
  LAB       FIBS       0       0               0        1
  ASGN      0          0       FIB1            0        5
  ASGN      1          0       FIB2            0        9
  ASGN      100        0       N               0        13
  ASGN      1          0       I               0        17
  LAB       .L6        0       0               0        21
  GT        I          N       .T7             0        25
  BT        .T7        0       .L8             0        29
  ASGN      FIB1       0       FIB             0        33
  ASGN      FIB2       0       FIB1            0        37
  ADD       FIB        FIB1    .T11            0        41
  ASGN      .T11       0       FIB2            0        45
  ADD       I          1       .T13            0        49
  ASGN      .T13       0       I               0        53
  BR        0          0       .L6             0        57
  LAB       .L8        0       0               0        61
```

```
                 R QUADS

OPERATOR   OPERAND1/   OPERAND2   RESULT
           CONDITION              LABEL
  LAB        FIBS        0          0
  RD         0           0          R6
  RD         1           0          R5
  RD         100         0          R4
  WT         R4          0          N
  WT         R5          0          I
  WT         R5          0          FIB2
  WT         R6          0          FIB1
  LAB        .L6         0          0
  RD         I           0          R6
  RD         N           0          R5
  GT         R6          R5         R1
  BT         R1          0          .L8
  RD         FIB1        0          R4
  RD         FIB2        0          R3
  ADD        R4          R3         R1
  RD         1           0          R2
  ADD        R6          R2         R5
  WT         R1          0          FIB2
  WT         R3          0          FIB1
  WT         R4          0          FIB
  WT         R5          0          I
  BR         0           0          .L6
  LAB        .L8         0          0
```

(a) FIBONACCI SERIES

```
            QUADS GENERATED

OPERATOR  OPERAND1/  OPERAND2 RESULT/   ****  RCD_NR  LOC_QUAD
          CONDITION           LABEL
  LAB     GRT_ELM      0      0                0        1
  ASGN    10           0      N                0        5
  SUBS    ARRAY        1      .T3              0        9
  ASGN    .T3          0      GRTST_ELMNT      0        13
  ASGN    1            0      INDEX            0        17
  ASGN    2            0      I                0        21
  LAB     .L7          0      0                0        25
  GT      I            N      .T8              0        29
  BT      .T8          0      .L9              0        33
  SUBS    ARRAY        I      .T10             0        37
  GT      GRTST_ELMNT .T10    .T11             0        41
  BT      .T11         0      LAB1             0        45
  SUBS    ARRAY        I      .T13             0        49
  ASGN    .T13         0      GRTST_ELMNT      0        53
  ASGN    I            0      INDEX            0        57
  LAB     LAB1         0      0                0        61
  ADD     I            1      .T17             0        65
  ASGN    .T17         0      I                0        69
  BR      0            0      .L7              0        73
  LAB     .L9          0      0                0        77
```

(b) GREATEST ELEMENT

```
                 R QUADS

OPERATOR   OPERAND1/    OPERAND2   RESULT
           CONDITION               LABEL
  LAB        GRT_ELM       0         0
  RD         10            0         R6
  RDAD       ARRAY         0         R5
  RD         1             0         R4
  ASL        R4            0         R1
  ADD        R5            R1        R1
  RDVR       R1            0         R1
  RD         2             0         R3
  WT         R1            0         GRTST_ELMNT
  WT         R3            0         I
  WT         R4            0         INDEX
  WT         R6            0         N
  LAB        .L7           0         0
  RD         I             0         R6
  RD         N             0         R5
  GT         R6            R5        R1
  BT         R1            0         .L9
  RDAD       ARRAY         0         R4
  ASL        R6            0         R1
  ADD        R4            R1        R1
  RDVR       R1            0         R1
  RD         GRTST_ELMNT   0         R3
  GT         R3            R1        R2
  BT         R2            0         LAB1
  ASL        R6            0         R1
  AND        R4            R1        R1
  RDVR       R1            0         R1
  WT         R1            0         GRTST_ELMNT
  WT         R6            0         INDEX
  LAB        LAB1          0         0
  RD         I             0         R6
  RD         1             0         R5
  ADD        R6            R5        R1
  WT         R1            0         I
  BR         0             0         .L7
  LAB        .L9           0         0
```

APPENDIX 7.3-2

```
                         QUADS GENERATED

OPERATOR  OPERAND1/   OPERAND2  RESULT/   ****   RCD_NR   LOC_QUAD
          CONDITION             LABEL
  LAB       PRIME        0        0                0         1
  ASGN      1            0        I                0         5
  LAB       .L3          0        0                0         9
  GT        I            256      .T4              0         13
  BT        .T4          0        .L5              0         17
  SUBL      A            I        .T6              0         21
  ASGN      1            0        .T6              0         25
  ADD       I            I        .T8              0         29
  ASGN      .T8          0        I                0         33
  BR        0            0        .L3              0         37
  LAB       .L5          0        0                0         41
  ASGN      2            0        I                0         45
  LAB       .L13         0        0                0         49
  GT        I            256      .T14             0         53
  BT        .T14         0        .L15             0         57
  SUBS      A            I        .T16             0         61
  EQ        .T16         0        .T17             0         65
  BT        .T17         0        LAB1             0         69
  MUL       2            I        .T19             0         73
  ASGN      .T19         0        K                0         77
  ASGN      K            0        J                0         81
  LAB       .L22         0        0                0         85
  GT        J            256      .T23             0         89
  BT        .T23         0        .L24             0         93
  SUBL      A            J        .T25             0         97
  ASGN      0            0        .T25             0         101
  ADD       J            I        .T27             0         105
  ASGN      .T27         0        J                0         109
  BR        0            0        .L22             0         113
  LAB       .L24         0        0                0         117
  LAB       LAB1         0        0                0         121
  ADD       J            I        .T32             0         125
  ASGN      .T32         0        J                0         129
  [illegible] 0          0        .L13             0         133
  L[illegible] .L15      0        0                0         137
```

```
                    B QUADS

OPERATOR  OPERAND1/   OPERAND2   RESULT
          CONDITION              LABEL
  LAB       PRIME        0         0
  RD        I            0         R6
  WT        R6           0         I
  LAB       .L3          0         0
  RD        I            0         R6
  RD        256          0         R5
  GT        R6           R5        R1
  BT        R1           0         .L5
  RDAD      A            0         R4
  ASL       R6           0         R1
  ADD       R4           R1        R1
  RD        1            0         R3
  WTAD      R3           0         R1
  ADD       R6           R3        R1
  WT        R1           0         I
  BR        0            0         .L3
  LAB       .L5          0         0
  RD        2            0         R6
  WT        R6           0         I
  LAB       .L13         0         0
  RD        I            0         R6
  RD        256          0         R5
  GT        R6           R5        R1
  BT        R1           0         .L15
  RDAD      A            0         R4
  ASL       R6           0         R1
  ADD       R4           R1        R1
  RDVR      R1           0         R1
  RD        0            0         R3
  EQ        R1           R3        R2
  BT        R2           0         LAB1
  RD        2            0         R2
  MUL       R2           R6        R1
  RD        J            0         R6
  RD        256          0         R5
  GT        R6           R5        R1
  BT        R1           0         .L24
  RDAD      A            0         R4
  ASL       R6           0         R1
  ADD       R4           R1        R1
  RD        0            0         R3
  WTAD      R3           0         R1
  RD        I            0         R2
  ADD       R6           R2        R1
  WT        R1           0         J
  BR        0            0         .L22
  LAB       .L24         0         0
  LAB       LAB1         0         0
  RD        J            0         R6
  RD        I            0         R5
  ADD       R6           R5        R1
  WT        R1           0         J
  BR        0            0         .L13
  LAB       .L15         0         0
```

(c) PRIME IDENTIFIER

APPENDIX 7.3-3

```
                QUADS GENERATED

OPERATOR  OPERAND1/  OPERAND2 RESULT/  ****  RCD_NR  LOC_QUAD
          CONDITION            LABEL
  LAB       SORT        0      0              0         1
  ASGN      0           0      SWITCH         0         5
  ASGN      0           0      K              0         9
  LAB       LAB2        0      0              0        13
  ADD       K           1      .T5            0        17
  ASGN      .T5         0      K              0        21
  ASGN      K           0      I              0        25
  LAB       .L8         0      0              0        29
  GT        I           9      .T9            0        33
  BT        .T9         0      .L10           0        37
  ADD       I           1      .T11           0        41
  ASGN      .T11        0      J              0        45
  SUBS      ARRAY       I      .T13           0        49
  SUBS      ARRAY       J      .T14           0        53
  LT        .T13        .T14   .T15           0        57
  BT        .T15        0      LAB1           0        61
  ASGN      1           0      SWITCH         0        65
  SUBS      ARRAY       I      .T18           0        69
  ASGN      .T18        0      TEMP           0        73
  SUBL      ARRAY       I      .T20           0        77
  SUBS      ARRAY       J      .T21           0        81
  ASGN      .T21        0      .T20           0        85
  SUBL      ARRAY       J      .T23           0        89
  ASGN      TEMP        0      .T23           0        93
  LAB       LAB1        0      0              0        97
  ADD       I           1      .T26           0       101
  ASGN      .T26        0      I              0       105
  BR        0           0      .L8            0       109
  LAB       .L10        0      0              0       113
  EQ        SWITCH      1      .T30           0       117
  BT        .T30        0      LAB2           0       121
```

(d) BUBBLE SORT

```
               R QUADS

OPERATOR   OPERAND1/   OPERAND2   RESULT
           CONDITION              LABEL
  LAB        SORT         0         0
  RD         0            0         R6
  WT         R6           0         K
  WT         R6           0         SWITCH
  LAB        LAB2         0         0
  RD         K            0         R6
  RD         1            0         R5
  ADD        R6           R5        R1
  WT         R1           0         I
  WT         R1           0         K
  LAB        .L8          0         0
  RD         I            0         R6
  RD         9            0         R5
  GT         R6           R5        R1
  BT         R1           0         .L10
  RD         1            0         R4
  ADD        R6           R4        R1
  RDAD       ARRAY        0         R3
  ASL        R6           0         R2
  ADD        R3           R2        R2
  RDVR       R2           0         R2
  ASL        R1           0         R4
  ADD        R3           R4        R4
  RDVR       R4           0         R4
  LT         R2           R4        R3
  WT         R1           0         J
  BT         R3           0         LAB1
  RD         1            0         R4
  RDAD       ARRAY        0         R3
  ASL        R6           0         R2
  ADD        R3           R2        R2
  RDVR       R2           0         R2
  ASL        R6           0         R1
  ADD        R3           R1        R1
  RD         J            0         R5
  ASL        R5           0         R6
  ADD        R3           R6        R6
  RDVR       R6           0         R6
  WTAD       R6           0         R1
  ASL        R5           0         R1
  ADD        R3           R1        R1
  WTAD       R2           0         R1
  WT         R2           0         TEMP
  WT         R4           0         SWITCH
  LAB        LAB1         0         0
  RD         I            0         R6
  RD         1            0         R5
  ADD        R6           R5        R1
  WT         R1           0         I
  BR         0            0         .L8
  LAB        .L10         0         0
  RD         SWITCH       0         R6
  RD         1            0         R5
  EQ         R6           R5        R1
  BT         R1           0         LAB2
```

APPENDIX 7.3-4

```
                    QUADS GENERATED

OPERATOR   OPERAND1/   OPERAND2 RESULT/   ****   RCD_NR   LOC_QUAD
           CONDITION            LABEL
   LAB       FILTER       0       0                 0         1
   SUBL      B            1       .T2               0         5
   ASGN      0            0       .T2               0         9
   ASGN      1            0       I                 0        13
   LAB       .L5          0       0                 0        17
   GT        I            N       .T6               0        21
   BT        .T6          0       .L7               0        25
   ADD       I            1       .T8               0        29
   ASGN      .T8          0       J                 0        33
   SUBL      A            I       .T10              0        37
   SUBS      A            I       .T11              0        41
   MUL       .T11         AT      .T12              0        45
   ASGN      .T12         0       .T10              0        49
   SUBL      B            J       .T14              0        53
   SUBS      B            I       .T15              0        57
   MUL       .T15         EXAT    .T16              0        61
   ASGN      .T16         0       .T14              0        65
   SUBS      A            I       .T18              0        69
   SUBS      B            J       .T19              0        73
   ADD       .T18         .T19    .T20              0        77
   ASGN      .T20         0       C                 0        81
   SUBL      B            J       .T22              0        85
   ASGN      C            0       .T22              0        89
   ADD       I            1       .T24              0        93
   ASGN      .T24         0       I                 0        97
   BR        0            0       .L5               0       101
   LAB       .L7          0       0                 0       105
```

(e) DIGITAL FILTER

```
                R QUADS

OPERATOR   OPERAND1/   OPERAND2   RESULT
           CONDITION              LABEL
   LAB       FILTER       0         0
   RDAD      B            0         R6
   RD        1            0         R5
   ASL       R5           0         R1
   ADD       R6           R1        R1
   RD        0            0         R4
   WTAD      R4           0         R1
   WT        R5           0         I
   LAB       .L5          0         0
   RD        I            0         R6
   RD        N            0         R5
   GT        R6           R5        R1
   BT        R1           0         .L7
   RD        J            0         R4
   ADD       R6           R4        R1
   RDAD      A            0         R3
   ASL       R6           0         R2
   ADD       R3           R2        R2
   ASL       R6           0         R4
   ADD       R3           R4        R4
   RDVR      R4           0         R4
   RD        AT           0         R3
   MUL       R4           R3        R5
   WTAD      R5           0         R2
   RDAD      B            0         R5
   ASL       R1           0         R2
   ADD       R5           R2        R2
   ASL       R6           0         R4
   ADD       R5           R4        R4
   RDVR      R4           0         R4
   RD        EXAT         0         R3
   MUL       R4           R3        R5
   WTAD      R5           0         R2
   RDAD      A            0         R5
   ASL       R6           0         R2
   ADD       R5           R2        R2
   RDVR      R2           0         R2
   RDAD      B            0         R4
   ASL       R1           0         R3
   ADD       R4           R3        R3
   RDVR      R3           0         R3
   ADD       R2           R3        R4
   RDAD      B            0         R3
   ASL       R1           0         R2
   ADD       R3           R2        R2
   WTAD      R4           0         R2
   RD        1            0         R2
   ADD       R6           R2        R3
   WT        R1           0         J
   WT        R3           0         I
   WT        R4           0         C
   BR        0            0         .L5
   LAB       .L7          0         0
```

APPENDIX 7.4-1 DEC PDP 11/45 MICROCODE REPRESENTATION OF TEST PROGRAMS

(a) FIBONACCI SERIES

GENERATED MICRO WORDS

LOC	CLK	CIR	WF	CH	CD	CRA	BUS	DAD	SPS	ALU	SBC	SBM	SDM	SBA	UBF	SRX	RIF	UPF
000	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	00000001
001	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	00000010
002	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00000011
003	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00000100
004	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00000101
005	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0110	00000110
006	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00000111
007	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00001000
010	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00001001
011	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0101	00001010
012	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00001011
013	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00001100
014	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00001101
015	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0100	00001110
016	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00001111
017	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00010000
020	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	00010001
021	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0100	00010010
022	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00010011
023	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00010100
024	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	00010101
025	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	04000	0001	0101	00010110
026	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00010111
027	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00011000
030	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	00011001
031	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0101	00011010
032	111	0	11	0	1	1	001	[illegible]	000	01001	0010	1111	10	1	00000	0001	0000	00011011
033	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00011100
034	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	00011101
035	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0110	00011110
036	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	00011111
037	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	00100000
040	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00100001
041	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00100010
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	01	0	00000	0001	0110	00100100
044	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00100101
045	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00100110
046	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00100111
047	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0101	00101000
050	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0101	00101001
051	100	0	00	0	1	0	000	1000	011	00110	0000	0000	00	0	00000	0001	0110	00101010
052	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00101011
053	100	0	00	0	1	0	000	0000	001	10011	0001	1111	00	0	01010	0000	0000	00101101
055	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	00101110
056	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	00110000
057	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	01010010
060	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00110001
061	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00110010
062	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00110011
063	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0100	00110100
064	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00110101
065	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00110110
066	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00110111
067	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0011	00111000
070	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0011	00111001
071	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0100	00111010
072	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00111011
073	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00111100
074	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00111101
075	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00111110
076	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0010	00111111
077	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0010	01000000
100	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0110	01000001
101	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0101	01000010
102	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	01000011
103	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	01000100
104	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	01000101
105	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0001	01000110
106	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	01000111
107	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	01001000
110	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	01001001
111	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0011	01001010
112	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	01001011
113	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	01001100
114	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	01001101
115	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0100	01001110
116	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	01001111
117	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	01010000
120	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	01010001
121	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0101	00011110
122	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	01010011
123	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	01010100

APPENDIX 7.4-2

(b) GREATEST ELEMENT - 1

```
                                           GENERATED MICRO WORDS

 L    C   C W  C C C  B    D    S    A     S    S   S  S   U     S    R      U
 U    L   I R  H D B  U    A    P    L     R    H   D  B   B     R    I      P
 C    K   R        A  S    D    S    U     C    M   M  A   F     X    F      F
000  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0000 00000001
001  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0000 00000010
002  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 00000011
003  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 00000100
004  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 00000101
005  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0110 00000110
006  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 00000111
007  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0101 00001000
010  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 00001001
011  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 00001010
012  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 00001011
013  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0100 00001100
014  100 0 00 0 1 0 000 0000 000 01100 0000 0000 00 0 00000 0001 0100 00001101
015  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 00001110
016  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0001 00001111
017  100 0 00 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0101 00010000
020  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 00010001
021  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0001 00010010
022  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0001 00010011
023  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 00010100
024  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 00010101
025  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 00010110
026  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0011 00010111
027  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 00011000
030  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 00011001
031  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 1000 00011010
032  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0001 00011011
033  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 00011100
034  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 00011101
035  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 1000 00011110
036  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0011 00011111
037  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 00100000
040  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 00100001
041  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 1000 00100[illegible]
[illegible]
043  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 00100100
044  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 00100101
045  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 1000 00100110
046  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0110 00100111
047  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0000 00101000
050  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0000 00101001
051  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 00101010
052  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 00101011
053  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 00101100
054  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0110 00101101
055  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 00101110
056  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 00101111
057  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 00110000
060  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0101 00110001
061  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0101 00110010
062  100 0 00 0 1 0 000 1000 011 00110 0000 0000 00 0 00000 0001 0110 00110011
063  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 00110100
064  100 0 00 0 1 0 000 0000 001 10011 0001 1111 00 0 01010 0000 0000 00110101
065  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 00110110
066  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 00111000
067  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 01101100
070  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 00111001
071  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0100 00111010
072  100 0 00 0 1 0 000 0000 000 01100 0000 0000 00 0 00000 0001 0110 00111011
073  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 00111100
074  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0001 00111101
075  100 0 00 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0100 00111110
076  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 00111111
077  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0001 01000000
100  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0001 01000001
101  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01000010
102  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 01000011
103  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 01000100
104  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0011 01000101
105  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0001 01000110
106  100 0 00 0 1 0 000 1000 011 00110 0000 0000 00 0 00000 0001 0011 01000111
107  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0010 01001000
110  100 0 00 0 1 0 000 0000 001 10011 0001 1111 00 0 01010 0000 0000 01001001
111  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 01001010
112  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 01001100
```

(b) GREATEST ELEMENT - 2

```
113  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 01011011
114  100 0 00 0 1 0 000 0000 000 01100 0000 0000 00 0 00000 0001 0110 01001101
115  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 01001110
116  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0001 01001111
117  100 0 00 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0100 01010000
120  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 01010001
121  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0001 01010010
122  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0001 01010011
123  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01010100
124  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 01010101
125  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 1000 01010110
126  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0001 01010111
127  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01011000
130  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 01011001
131  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 1000 01011010
132  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0110 01011011
133  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0000 01011100
134  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0000 01011101
135  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01011110
136  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 01011111
137  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 01100000
140  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0110 01100001
141  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01100010
142  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 01100011
143  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 01100100
144  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0101 01100101
145  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0101 01100110
146  100 0 00 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0110 01100111
147  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 01101000
150  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01101001
151  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 01101010
152  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 1000 01101011
153  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0001 00100111
154  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0000 01101101
155  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0000 01101110
```

APPENDIX 7.4-4

(c) PRIME IDENTIFIER 1

```
                                  GENERATED MICRO WORDS

L    C   C W  C C C R   D    S   A     S    S    S  S U     S    R    U
O    L   I R  B D H O   A    P   L     B    H    D  B B     R    I    P
C    K   R         A S   D    S   U     C    M    M  A F     X    F    F
000  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0000 00000001
001  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0000 00000010
002  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 00000011
003  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 00000100
004  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 00000101
005  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0110 00000110
006  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 00000111
007  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 00001000
010  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 1000 00001001
011  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0110 00001010
012  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0000 00001011
013  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0000 00001100
014  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 00001101
015  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 00001110
016  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 00001111
017  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0110 00010000
020  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 00010001
021  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 00010010
022  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 00010011
023  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0101 00010100
024  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0101 00010101
025  100 0 00 0 1 0 000 1000 011 00110 0000 0000 00 0 00000 0001 0110 00010110
026  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 00010111
027  100 0 00 0 1 0 000 0000 001 10011 0001 1111 00 0 01010 0000 0000 00011001
031  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 00011010
032  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 00011100
033  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 00110000
034  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 00011101
035  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0100 00011110
036  100 0 00 0 1 0 000 0000 000 01100 0000 0000 00 0 00000 0001 0110 00011111
037  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 00100000
040  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0001 00100001
041  100 0 00 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0100 00100010
042  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 00100011

[illegible] 111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 00100100
044  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 00100101
045  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 00100110
046  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0011 00100111
047  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 0011 00101000
050  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0001 00101001
051  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0011 00101010
052  100 0 00 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0110 00101011
053  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 00101100
054  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 00101101
055  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 00101110
056  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 1000 00101111
057  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0001 00001010
060  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0000 00110001
061  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0000 00110010
062  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 00110011
063  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 00110100
064  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 00110101
065  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0110 00110110
066  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 00110111
067  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 00111000
070  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 1000 00111001
071  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0110 00111010
072  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0000 00111011
073  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0000 00111100
074  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 00111101
075  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 00111110
076  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 00111111
077  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0110 01000000
100  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01000001
101  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 01000010
102  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 01000011
103  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0101 01000100
104  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0101 01000101
105  100 0 00 0 1 0 000 1000 011 00110 0000 0000 00 0 00000 0001 0110 01000110
106  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 01000111
107  100 0 00 0 1 0 000 0000 001 10011 0001 1111 00 0 01010 0000 0000 01001001
111  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 01001010
112  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 01001100
113  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 10110101
114  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01001101
```

APPENDIX 7.4-5

(c) PRIME IDENTIFIER - 2

```
115  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0100 01001110
116  100 0 00 0 1 0 000 0000 000 01100 0000 0000 00 0 00000 0001 0110 01001111
117  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 01010000
120  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0001 01010001
121  100 0 00 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0100 01010010
122  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 01010011
123  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0001 01010100
124  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0001 01010101
125  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01010110
126  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 01010111
127  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 01011000
130  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0011 01011001
131  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0011 01011010
132  100 0 00 0 1 0 000 1000 011 00110 0000 0000 00 0 00000 0001 0001 01011011
133  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0010 01011100
134  100 0 00 0 1 0 000 0000 001 10011 0001 1111 00 0 01010 0000 0000 01011101
135  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 01011110
136  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 01100000
137  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 10100100
140  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01100001
141  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 01100010
142  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 01100011
143  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0010 01100100
144  110 0 11 0 1 0 000 0000 000 11010 1111 1111 10 0 00000 0001 1000 01100101
145  110 0 11 0 1 0 000 0000 000 01100 0000 0000 10 0 00000 0001 1000 01100110
146  110 0 11 0 1 0 000 0000 000 01100 0000 0000 10 0 00000 0001 1000 01100111
147  110 0 11 0 1 0 000 0000 000 00011 0000 0000 10 0 00000 0001 0001 01101000
150  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0010 01101010
152  110 0 11 0 1 0 000 0000 100 00000 0000 0000 10 0 10011 0001 0110 01101011
153  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 01101100
154  100 0 11 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0001 01101101
155  100 0 11 0 1 0 000 0000 001 00110 0001 1111 10 0 01010 0001 1000 01101111
157  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 01101010
160  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 01110001
161  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01110010
162  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 01110011
163  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 1000 01110100
164  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0001 [illegible]
170  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0001 01111001
171  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0000 01111010
172  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0000 01111011
173  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01111100
174  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 01111101
175  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 01111110
176  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0110 01111111
177  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 10000000
200  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 10000001
201  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 10000010
202  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0101 10000011
203  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0101 10000100
204  100 0 00 0 1 0 000 1000 011 00110 0000 0000 00 0 00000 0001 0110 10000101
205  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 10000110
206  100 0 00 0 1 0 000 0000 001 10011 0001 1111 00 0 01010 0000 0000 10000111
207  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 10001000
210  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 10001010
211  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 10100010
212  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 10001011
213  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0100 10001100
214  100 0 00 0 1 0 000 0000 000 01100 0000 0000 00 0 00000 0001 0110 10001101
215  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 10001110
216  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0001 10001111
217  100 0 00 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0100 10010000
220  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 10010001
221  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 10010010
222  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 10010011
223  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 10010100
224  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0011 10010101
225  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 0011 10010110
226  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0001 10010111
227  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 10011000
230  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 10011001
231  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 10011010
232  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0010 10011011
233  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0010 10011100
234  100 0 00 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0110 10011101
235  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 10011110
236  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 10011111
```

APPENDIX 7.4-6

(c) PRIME IDENTIFIER ← 3

```
237   010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 10100000
240   010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 1000 10100001
241   101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0001 01111001
242   011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0000 10100011
243   010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0000 10100100
244   011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0000 10100101
245   010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0000 10100110
246   111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 10100111
247   010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 10101000
250   011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 10101001
251   010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0110 10101010
252   111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 10101011
253   010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 10101100
254   011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 10101101
255   010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0101 10101110
256   010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0101 10101111
257   100 0 00 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0110 10110000
260   010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 10110001
261   111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 10110010
262   010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 10110011
263   010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 1000 10110100
264   101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0001 00111010
265   011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0000 10110110
266   010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0000 10110111
```

APPENDIX 7.4-7

(d) BUBBLE SORT - 1

```
                         GENERATED MICRO WORDS

L     C    C  W   C  C  C  R    D     S    A      S     S     S   S  U      S     R     U
U     L    I  R   B  D  R  U    A     P    L      R     H     D   B  B      R     I     P
C     K    R            A  S    D     S    U      C     M     M   A  F      X     F     F
000   011  0  00  0  0  1  001  0000  000  00000  0000  0000  00  1  00000  0001  0000  00000001
001   010  0  11  0  0  0  000  0000  000  00000  0000  0000  01  0  00000  0001  0000  00000010
002   111  0  11  0  1  1  001  0000  000  01001  0010  1111  10  1  00000  0001  0000  00000011
003   010  0  11  0  0  0  000  0000  000  00000  0000  0000  01  0  00000  0001  1000  00000100
004   011  0  00  0  0  1  001  0000  000  00000  0000  0000  00  1  00000  0001  1000  00000101
005   010  0  11  0  0  0  000  0000  000  00000  0000  0000  01  0  00000  0001  0110  00000110
006   111  0  11  0  1  1  001  0000  000  01001  0010  1111  10  1  00000  0001  0000  00000111
007   010  0  11  0  0  0  000  0000  000  00000  0000  0000  01  0  00000  0001  1000  00001000
010   010  0  00  0  0  1  000  0000  000  00000  0000  0000  00  1  00000  0001  1000  00001001
011   101  0  00  0  1  0  101  0000  000  00000  0000  0000  00  0  00000  0001  0110  00001010
012   111  0  11  0  1  1  001  0000  000  01001  0010  1111  10  1  00000  0001  0000  00001011
013   010  0  11  0  0  0  000  0000  000  00000  0000  0000  01  0  00000  0001  1000  00001100
014   010  0  00  0  0  1  000  0000  000  00000  0000  0000  00  1  00000  0001  1000  00001101
015   101  0  00  0  1  0  101  0000  000  00000  0000  0000  00  0  00000  0001  0110  00001110
016   011  0  00  0  0  1  001  0000  000  00000  0000  0000  00  1  00000  0001  0000  00001111
017   010  0  11  0  0  0  000  0000  000  00000  0000  0000  01  0  00000  0001  0000  00010000
020   111  0  11  0  1  1  001  0000  000  01001  0010  1111  10  1  00000  0001  0000  00010001
021   010  0  11  0  0  0  000  0000  000  00000  0000  0000  01  0  00000  0001  1000  00010010
022   011  0  00  0  0  1  001  0000  000  00000  0000  0000  00  1  00000  0001  1000  00010011
023   010  0  11  0  0  0  000  0000  000  00000  0000  0000  01  0  00000  0001  0110  00010100
024   111  0  11  0  1  1  001  0000  000  01001  0010  1111  10  1  00000  0001  0000  00010101
025   010  0  11  0  0  0  000  0000  000  00000  0000  0000  01  0  00000  0001  1000  00010110
026   011  0  00  0  0  1  001  0000  000  00000  0000  0000  00  1  00000  0001  1000  00010111
027   010  0  11  0  0  0  000  0000  000  00000  0000  0000  01  0  00000  0001  0101  00011000
030   010  0  00  1  0  0  000  0000  000  00000  0000  0000  00  0  00000  0001  0101  00011001
031   100  0  00  0  1  0  000  0000  000  01001  0000  0000  00  0  00000  0001  0110  00011010
032   010  0  11  0  0  0  000  0000  000  00000  0000  0000  10  0  00000  0001  0001  00011011
033   111  0  11  0  1  1  001  0000  000  01001  0010  1111  10  1  00000  0001  0000  00011100
034   010  0  11  0  0  0  000  0000  000  00000  0000  0000  01  0  00000  0001  1000  00011101
035   010  0  00  0  0  1  000  0000  000  00000  0000  0000  00  1  00000  0001  1000  00011110
036   101  0  00  0  1  0  101  0000  000  00000  0000  0000  00  0  00000  0001  0001  00011111
037   111  0  11  0  1  1  001  0000  000  01001  0010  1111  10  1  00000  0001  0000  00100000
040   010  0  11  0  0  0  000  0000  000  00000  0000  0000  01  0  00000  0001  1000  00100001
041   010  0  00  0  0  1  000  0000  000  00000  0000  0000  00  1  00000  0001  1000  00100010
042   101  0  00  0  1  0  101  0000  000  00000  0000  0000  00  0  00000  0001  0001  00100011
043   011  0  00  0  0  1  001  0000  000  00000  0000  0000  00  1  00000  0001  0000  00100100
044   010  0  11  0  0  0  000  0000  000  00000  0000  0000  01  0  00000  0001  0000  00100101
045   111  0  11  0  1  1  001  0000  000  01001  0010  1111  10  1  00000  0001  0000  00100110
046   010  0  11  0  0  0  000  0000  000  00000  0000  0000  01  0  00000  0001  1000  00100111
047   011  0  00  0  0  1  001  0000  000  00000  0000  0000  00  1  00000  0001  1000  00101000
050   010  0  11  0  0  0  000  0000  000  00000  0000  0000  01  0  00000  0001  0110  00101001
051   111  0  11  0  1  1  001  0000  000  01001  0010  1111  10  1  00000  0001  0000  00101010
052   010  0  11  0  0  0  000  0000  000  00000  0000  0000  01  0  00000  0001  1000  00101011
053   011  0  00  0  0  1  001  0000  000  00000  0000  0000  00  1  00000  0001  1000  00101100
054   010  0  11  0  0  0  000  0000  000  00000  0000  0000  01  0  00000  0001  0101  00101101
055   010  0  00  1  0  0  000  0000  000  00000  0000  0000  00  0  00000  0001  0101  00101110
056   100  0  00  0  1  0  000  1000  011  00110  0000  0000  00  0  00000  0001  0110  00101111
057   010  0  11  0  0  0  000  0000  000  00000  0000  0000  10  0  00000  0001  0001  00110000
060   100  0  00  0  1  0  000  0000  001  10011  0001  1111  00  0  01010  0000  0000  00110001
061   010  0  00  0  0  0  000  0000  000  00000  0000  0000  00  0  00000  0000  0000  00110010
062   010  0  00  0  0  0  000  0000  000  00000  0000  0000  00  0  00000  0000  0000  00110100
063   010  0  00  0  0  0  000  0000  000  00000  0000  0000  00  0  00000  0000  0000  10010101
064   111  0  11  0  1  1  001  0000  000  01001  0010  1111  10  1  00000  0001  0000  00110101
065   010  0  11  0  0  0  000  0000  000  00000  0000  0000  01  0  00000  0001  1000  00110110
066   011  0  00  0  0  1  001  0000  000  00000  0000  0000  00  1  00000  0001  1000  00110111
067   010  0  11  0  0  0  000  0000  000  00000  0000  0000  01  0  00000  0001  0100  00111000
070   010  0  00  1  0  0  000  0000  000  00000  0000  0000  00  0  00000  0001  0100  00111001
071   100  0  00  0  1  0  000  0000  000  01001  0000  0000  00  0  00000  0001  0110  00111010
072   010  0  11  0  0  0  000  0000  000  00000  0000  0000  10  0  00000  0001  0001  00111011
073   111  0  11  0  1  1  001  0000  000  01001  0010  1111  10  1  00000  0001  0000  00111100
074   010  0  11  0  0  0  000  0000  000  00000  0000  0000  01  0  00000  0001  0011  00111101
075   100  0  00  0  1  0  000  0000  000  01100  0000  0000  00  0  00000  0001  0110  00111110
076   010  0  11  0  0  0  000  0000  000  00000  0000  0000  10  0  00000  0001  0010  00111111
077   010  0  00  1  0  0  000  0000  000  00000  0000  0000  00  0  00000  0001  0010  01000000
100   100  0  00  0  1  0  000  0000  000  01001  0000  0000  00  0  00000  0001  0011  01000001
101   010  0  11  0  0  0  000  0000  000  00000  0000  0000  10  0  00000  0001  0010  01000010
102   011  0  00  0  0  1  001  0000  000  00000  0000  0000  00  1  00000  0001  0010  01000011
103   010  0  11  0  0  0  000  0000  000  00000  0000  0000  01  0  00000  0001  0010  01000100
104   100  0  00  0  1  0  000  0000  000  01100  0000  0000  00  0  00000  0001  0001  01000101
105   010  0  11  0  0  0  000  0000  000  00000  0000  0000  10  0  00000  0001  0100  01000110
106   010  0  00  1  0  0  000  0000  000  00000  0000  0000  00  0  00000  0001  0100  01000111
107   100  0  00  0  1  0  000  0000  000  01001  0000  0000  00  0  00000  0001  0011  01001000
110   010  0  11  0  0  0  000  0000  000  00000  0000  0000  10  0  00000  0001  0100  01001001
111   011  0  00  0  0  1  001  0000  000  00000  0000  0000  00  1  00000  0001  0100  01001010
112   010  0  11  0  0  0  000  0000  000  00000  0000  0000  01  0  00000  0001  0100  01001011
```

APPENDIX 7.4-8

(d) BUBBLE SORT 2

```
113  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0010 01001100
114  100 0 00 0 1 0 000 1000 011 00110 0000 0000 00 0 00000 0001 0100 01001101
115  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0011 01001110
116  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01001111
117  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 01010000
120  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 1000 01010001
121  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0001 01010010
122  100 0 00 0 1 0 000 0000 001 10011 0001 1111 00 0 01010 0000 0000 01010011
123  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 01010100
124  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 01010110
125  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 10000100
126  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01010111
127  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 01011000
130  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 01011001
131  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0100 01011010
132  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01011011
133  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0011 01011100
134  100 0 00 0 1 0 000 0000 000 01100 0000 0000 00 0 00000 0001 0110 01011101
135  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0010 01011110
136  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0010 01011111
137  100 0 00 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0011 01100000
140  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0010 01100001
141  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0010 01100010
142  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0010 01100011
143  100 0 00 0 1 0 000 0000 000 01100 0000 0000 00 0 00000 0001 0110 01100100
144  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 01100101
145  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0001 01100110
146  100 0 00 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0011 01100111
147  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 01101000
150  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01101001
151  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 01101010
152  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 01101011
153  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0101 01101100
154  100 0 00 0 1 0 000 0000 000 01100 0000 0000 00 0 00000 0001 0101 01101101
155  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0110 01101110
156  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0110 01101111
157  100 0 00 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0011 01110000
160  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0110 01110001
161  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0110 01110010
162  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0110 01110011
163  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 0110 01110100
164  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0001 01110101
165  100 0 00 0 1 0 000 0000 000 01100 0000 0000 00 0 00000 0001 0101 01110110
166  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 01110111
167  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0001 01111000
170  100 0 00 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0011 01111001
171  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 01111010
172  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 0010 01111011
173  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0001 01111100
174  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01111101
175  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 01111110
176  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 1000 01111111
177  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0010 10000000
200  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 10000001
201  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 10000010
202  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 1000 10000011
203  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0100 10000100
204  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0000 10000101
205  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0000 10000110
206  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 10000111
207  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 10001000
210  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 10001001
211  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0110 10001010
212  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 10001011
213  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 10001100
214  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 10001101
215  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0101 10001110
216  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0101 10001111
217  100 0 00 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0110 10010000
220  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 10010001
221  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 10010010
222  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 10010011
223  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 1000 10010100
224  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0001 00100011
225  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0000 10010110
226  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0000 10010111
227  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 10011000
230  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 10011001
231  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 10011010
232  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0110 10011011
233  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 10011100
234  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 10011101
235  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 10011110
236  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0101 10011111
237  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0101 10100000
240  100 0 00 0 1 0 000 1000 011 00110 0000 0000 00 0 00000 0001 0110 10100001
241  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 10100010
242  100 0 00 0 1 0 000 0000 001 10011 0001 1111 00 0 01010 0000 0000 10100011
243  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 10100100
244  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 10100110
245  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 00001110
```

APPENDIX 7.4-9

(e) DIGITAL FILTER - 1

GENERATED MICRO WORDS

LOC	CLK	CTR	WP	CA	CD	CRA	RUS	OAD	SPS	ALU	SBC	SBM	SDM	SBA	UHF	SRX	RIF	UPF
000	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	00000001
001	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	00000010
002	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00000011
003	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0110	00000100
004	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00000101
005	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00000110
006	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00000111
007	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0101	00001000
010	100	0	00	0	1	0	000	0000	000	01100	0000	0000	00	0	00000	0001	0101	00001001
011	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00001010
012	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0001	00001011
013	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0110	00001100
014	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00001101
015	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00001110
016	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00001111
017	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00010000
020	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0100	00010001
021	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	0100	00010010
022	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0001	00010011
023	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00010100
024	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00010101
025	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	00010110
026	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0101	00010111
027	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	00011000
030	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	00011001
031	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00011010
032	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00011011
033	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00011100
034	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0110	00011101
035	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00011110
036	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00011111
037	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00100000
040	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0101	00100001
041	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0101	00100010
042	100	0	00	0	1	0	000	1000	011	00110	0000	0000	00	0	00000	0001	0110	00100011
043	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00100100
044	100	0	00	0	1	0	000	0000	001	10011	0001	1111	00	0	01010	0000	0000	00100101
045	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	00100110
046	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	00101000
047	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	10100000
050	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00101001
051	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00101010
052	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00101011
053	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0100	00101100
054	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0100	00101101
055	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0110	00101110
056	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00101111
057	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00110000
060	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0011	00110001
061	100	0	00	0	1	0	000	0000	000	01100	0000	0000	00	0	00000	0001	0110	00110010
062	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0010	00110011
063	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0010	00110100
064	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0011	00110101
065	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0010	00110110
066	100	0	00	0	1	0	000	0000	000	01100	0000	0000	00	0	00000	0001	0110	00110111
067	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0100	00111000
070	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0100	00111001
071	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0011	00111010
072	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0100	00111011
073	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0100	00111100
074	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0100	00111101
075	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00111110
076	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00111111
077	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	01000000
100	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0011	01000001
101	110	0	11	0	1	0	000	0000	000	11010	1111	1111	10	0	00000	0001	1000	01000010
102	110	0	11	0	1	0	000	0000	000	01100	0000	0000	10	0	00000	0001	1000	01000011
103	110	0	11	0	1	0	000	0000	000	01100	0000	0000	10	0	00000	0001	1000	01000100
104	110	0	11	0	1	0	000	0000	000	00011	0000	0000	10	0	00000	0001	0101	01000101
105	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0100	01000110
106	110	0	11	0	1	0	000	0000	100	00000	0000	0000	10	0	10011	0001	0011	01000111
107	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	01001000
110	100	0	11	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0101	01001001
111	100	0	11	0	1	0	000	0000	001	00110	0001	1111	10	0	01010	0001	1000	01001011
113	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	01000110

(e) DIGITAL FILTER - 2

```
114  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 01001101
115  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 0101 01001110
116  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0010 01001111
117  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01010000
120  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0101 01010001
121  100 0 00 0 1 0 000 0000 000 01100 0000 0000 00 0 00000 0001 0001 01010010
122  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0010 01010011
123  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0010 01010100
124  100 0 00 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0101 01010101
125  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0010 01010110
126  100 0 00 0 1 0 000 0000 000 01100 0000 0000 00 0 00000 0001 0110 01010111
127  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0100 01011000
130  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0100 01011001
131  100 0 00 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0101 01011010
132  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0100 01011011
133  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0100 01011100
134  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0100 01011101
135  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01011110
136  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 01011111
137  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 01100000
140  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0011 01100001
141  110 0 11 0 1 0 000 0000 000 11010 1111 1111 10 0 00000 0001 1000 01100010
142  110 0 11 0 1 0 000 0000 000 01100 0000 0000 10 0 00000 0001 1000 01100011
143  110 0 11 0 1 0 000 0000 000 01100 0000 0000 10 0 00000 0001 1000 01100100
144  110 0 11 0 1 0 000 0000 000 00011 0000 0000 10 0 00000 0001 0101 01100101
145  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0100 01100110
146  110 0 11 0 1 0 000 0000 100 00000 0000 0000 10 0 10011 0001 0011 01100111
147  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 01101000
150  100 0 11 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0101 01101001
151  100 0 11 0 1 0 000 0000 001 00110 0001 1111 10 0 01010 0001 1000 01101011
153  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 01100110
154  010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 01101101
155  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 0101 01101110
156  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0010 01101111
157  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01110000
160  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0101 01110001
161  100 0 00 0 1 0 000 0000 000 01100 0000 0000 00 0 00000 0001 0110 01110010
162  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0010 01110011
163  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0010 01110100
164  100 0 00 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0101 01110101
165  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0010 01110110
166  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0010 01110111
167  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0010 01111000
170  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01111001
171  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0100 01111010
172  100 0 00 0 1 0 000 0000 000 01100 0000 0000 00 0 00000 0001 0001 01111011
173  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0011 01111100
174  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0011 01111101
175  100 0 00 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0100 01111110
176  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0011 01111111
177  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0011 10000000
200  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0011 10000001
201  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0011 10000010
202  100 0 00 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0010 10000011
203  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0100 10000100
204  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 10000101
205  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0011 10000110
206  100 0 00 0 1 0 000 0000 000 01100 0000 0000 00 0 00000 0001 0001 10000111
207  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0010 10001000
210  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0010 10001001
211  100 0 00 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0011 10001010
212  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0010 10001011
213  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 0100 10001100
214  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0010 10001101
215  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 10001110
216  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 10001111
217  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 10010000
220  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0010 10010001
221  010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0010 10010010
222  100 0 00 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0110 10010011
223  010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0011 10010100
224  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 10010101
225  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 10010110
226  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 1000 10010111
227  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0001 10011000
230  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 10011001
231  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 10011010
232  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 1000 10011011
233  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0011 10011100
234  111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 10011101
235  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 10011110
236  010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 1000 10011111
237  101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0100 00010111
240  011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0000 10100001
241  010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0000 10100010
```

7.5 (a) R. QUAD PROCESSOR PROCEDURES

The R. Quad Processor has 7 procedures. The calling sequence is shown below. Following this is a brief description of each procedure.

Procedure Name	Called By	Parameter	Return
ALLOC_REG_OP	ASGN QUAD Funct. QUADS SUBS QUAD SUBL QUAD BT/BF QUADS	Variable	Register Number
ALLOC_REG_RES	Funct. QUADS SUBS QUAD SUBL QUAD	Variable	Register Number
DEALLOCATE	ALLOC_REG_OP ALLOC_REG_RES		Register Number
DEALLOC_TEMP	ASGN QUAD Funct QUADS SUBS QUAD SUBL QUAD BT/BF QUADS	Variable Register Number	Register Number
FIND_OP	ALLOC_REG_OP	Variable	Register Number
FIND_RES	ALLOC_REG_RES ASGN_QUAD	Variable	Register Number
WRITE_REG	BR QUAD LAB QUAD BT/BF QUAD		

The procedure descriptions are provided below.

Alloc_Reg_OP: Procedure (Variable)

1. Reg=Find_OP (Variable)
   (Value of Reg is one of the following:
   a. register number which the variable is already assigned to.
   b. register number which the variable is to be assigned to.
   c. -1)
2. If Reg=-1, then Reg=Deallocate.
   (routine Deallocate returns a register number to be allocated to the variable, see explanation in Deallocate routine)

3. If RDFLAG is set to 'NO', then go to 8
   (RDFLAG is set by Find_OP routine to indicate if the variable
   is already assigned to the register)
   Else, go to 4.

4. Put the variable in the list of variables assigned to Reg and set
   its STATUS entry to 'ALLOCATED' and CHANGE entry to 'NO', set other
   entries accordingly, go to 5.

5. If the quad under process is a SUBS or SUBL quad, then go to 6.

   Else, go to 7.

6. Generate a RDAD register quad and go to 8.

   Quad sets Reg to address of the variable

7. Generate a RD register quad, and go to 8

   Reads value of the variable into Reg.

8. RETURN with value of Reg.

Alloc_Reg_Res: Procedure (Variable)

1. Reg=Find_Res(Variable)
   (Value of Reg is one of the following
   a. register number which the variable is already assigned to.

   b. register number which the variable is to be assigned to.

   c. -1)

2. If Reg= -1, then Reg=Deallocate.
   (routine Deallocate returns a register number to be allocated
   to the variable, see explanation on Deallocate routine)

3. If RDFLAG is set to 'NO' then go to 5
   (RDFLAG is set by Find_Res routine to indicate if the variable
   is already assigned to the register)

   Else, go to 4.

4. Put the variable in the list of variables assigned to Reg, set
   its STATUS entry to 'ALLOCATED' and CHANGE entry to "YES", and
   set other entries accordingly.

5. RETURN with value of Reg.

Deallocate: Procedure

(The routine will be called only in the case no register is available to be allocated to a variable.
The routine searches register table and selects a register on the following basis, notice that the given basis are in the order of importance attached to them in coding the routine.

1. Least number of temporary variables are assigned to the selected register.

2. Least number of variables are assigned to the selected register.

3. The longest time after the last reference to the selected register.

After selection of the register following steps are re-executed)

1. Search in the list of variables assigned to the register and generate a WT register quad for those variables whose CHANGE entry is set to 'YES'.
   (the generated register quad writes
   contents of Reg into memory location Var)

2. RETURN with value of register number.

Dealloc_Temp: Procedure (Variable, Register)

(If the given variable is a temporary variable, then it will be deleted from list of variables assigned to the register)

1. If the given variable is a temporary variable, then go to 2.

   Else, RETURN.

2. Delete the variable from list of variables assigned to the register.

3. If the deleted variable was the only variable assigned to the register, then go to 4.

   Else, RETURN.

4. Set STATUS entry of the register to 'FREE' and RETURN.

Find_OP: Procedure (Variable)
(returns a register number or -1, and sets RDFLAG)

1. Search list of variables assigned to all registers for the given variable.

2. If the variable is found, then go to 3.

   Else, go to 4.

3. Set RDFLAG to 'NO' and RETURN with value of register number which the variable is already assigned to.

4. Set READFLAG to 'YES'.

5. Search register table for a register with STATUS entry equal to 'FREE'.

6. If the search is successful, then RETURN with value of the register number.

   Else, RETURN with value of -1.

Find_Res: Procedure (Variable)
(returns a register number or -1 and sets READFLAG)

1. Set READFLAG to 'YES'.

1. Search list of variables assigned to all registers for the given variable.

2. If the variable is found, then go to 3.

   Else, go to 4.

3. If the variable is the only variable assigned to the register then set the CHANGE ENTRY OF THE VARIABLE TO'YES' and READFLAG to 'NO', and RETURN.

   Else, delete the variable from list of variables assigned to the register and go to 4.

4. Search register table for a register with STATUS entry equal to 'FREE'.

5. If the search is successful then RETURN with value of the register number.

   Else, RETURN with value of -1.

Write_Regs: Procedure

(the Procedure goes through list of variables assigned to all registers, and for those variables whose CHANGE entry is 'YES', generates a WT register quad, the generated register quads are
WT Reg Ø Var
which write contents of Reg in memory location associated to Var)

7.5 (b) <u>R QUAD GENERATORS</u>

The generators for the six R QUAD types are shown below.

1) Functional R QUADS

```
ADD  A  B  C
SUB  A  B  C
GT   A  B  C
LT   A  B  C
EQ   A  B  C
```

```
1. Reg 1 =Alloc_Reg_OP(A)
      (Reg 1 is register number which variable A is assigned to)

2. Reg 2=Alloc_Reg_OP(B)
      (Reg 2 is a register number which variable B is assigned tc)

3. Reg 3=Alloc_Reg_Res(C)
      (Reg 3 is a register number which variable C is assigned to)

4. Generate a register quad accordingly
      (the register quad will be one of the following:
      ADD  Reg 1  Reg 2  Reg 3
      SUB  Reg 1  Reg 2  Reg 3
      GT   Reg 1  Reg 2  Reg 3
      LT   Reg 1  Reg 2  Reg 3
      EQ   Reg 1  Reg 2  Reg 3)

5. Call Dealloc_Temp(A, Reg 1)
   Call Dealloc_Temp(B, Reg 2)
      (if A or B or both are temporary variables, then they will be
      deleted from list variables associated to their registers)
```

2) Assign R QUAD

```
ASGN  A  Ø  B

1. Reg 1=Alloc_Reg_OP(A);
      (Reg 1 is register number which variable A is assigned to)

2. Reg 2=Find_Res(B)
      (Reg 2 is a register number which variable B or address of
      variable B is assigned to)

3. If ADDRESS entry of Reg 2 is'YES', then generate a WTAD register
   quad, and go to 4.
      (generated register quad is
      WTAD  Reg 1  Ø  Reg 2
      the register quad writes contents of Reg 1 into location given
      by Reg 2)

   Else, insert variable B into list of variables assigned to Reg 1, set
   CHANGE entry of variable B to 'YES', set other entries accordingly,
   go to 4.
```

4. Delete variable B from list of variables assigned to Reg 2.

5. Call Dealloc_Temp(A, Reg 1)
   (Dealloc_Temp is a routine to delete variable A from Reg 2, if
   A is a temporary variable)

3) Array R QUADS

```
SUBS  A  I  C
SUBL  A  I  C
```

1. Reg 1= Alloc_Reg_OP(A)
   (Reg 1 is a register number to which address of first word of
   array A is assigned)

2. Reg 2= Alloc_Reg_OP(I)
   (Reg 2 is a register number to which variable I is assigned)

3. Reg 3= Alloc_Reg_Res(C)
   (Reg 3 is a register number to which variable C is assigned)

4. Generate an ASL register quad.
   (generated register quad is
   ASL Reg 2 Ø Reg 3
   the register quad sets Reg 3 to contents of Reg 2 shifted to
   left by one position)

5. Generate an ADD register quad.
   (generated register quad is
   ADD Reg 1 Reg 3 Reg 3
   the register quad adds contents of registers Reg 1 and Reg 3
   and sets Reg 3 to the result of addition, after the addition
   Reg 3 contains address of A(I))

6. If the quad under process is SUBS, then go to 7.

   Else, go to 8.
   (in this case it is a SUBL quad)

7. Generate a RDAD register quad, go to 9.
   (generated register quad is
   RDAD Reg 3 Ø Reg 3
   the register quad sets Reg 3 to contents of location given by
   Reg 3.)

8. Set ADDRS entry of Reg 3 to 'YES', go to 9.

9. Call Dealloc_Temp(I, Reg 2)
   (if I is a temporary variable, then it will be deleted from list of
   variables assigned to Reg 2)

4) Branch R QUADS

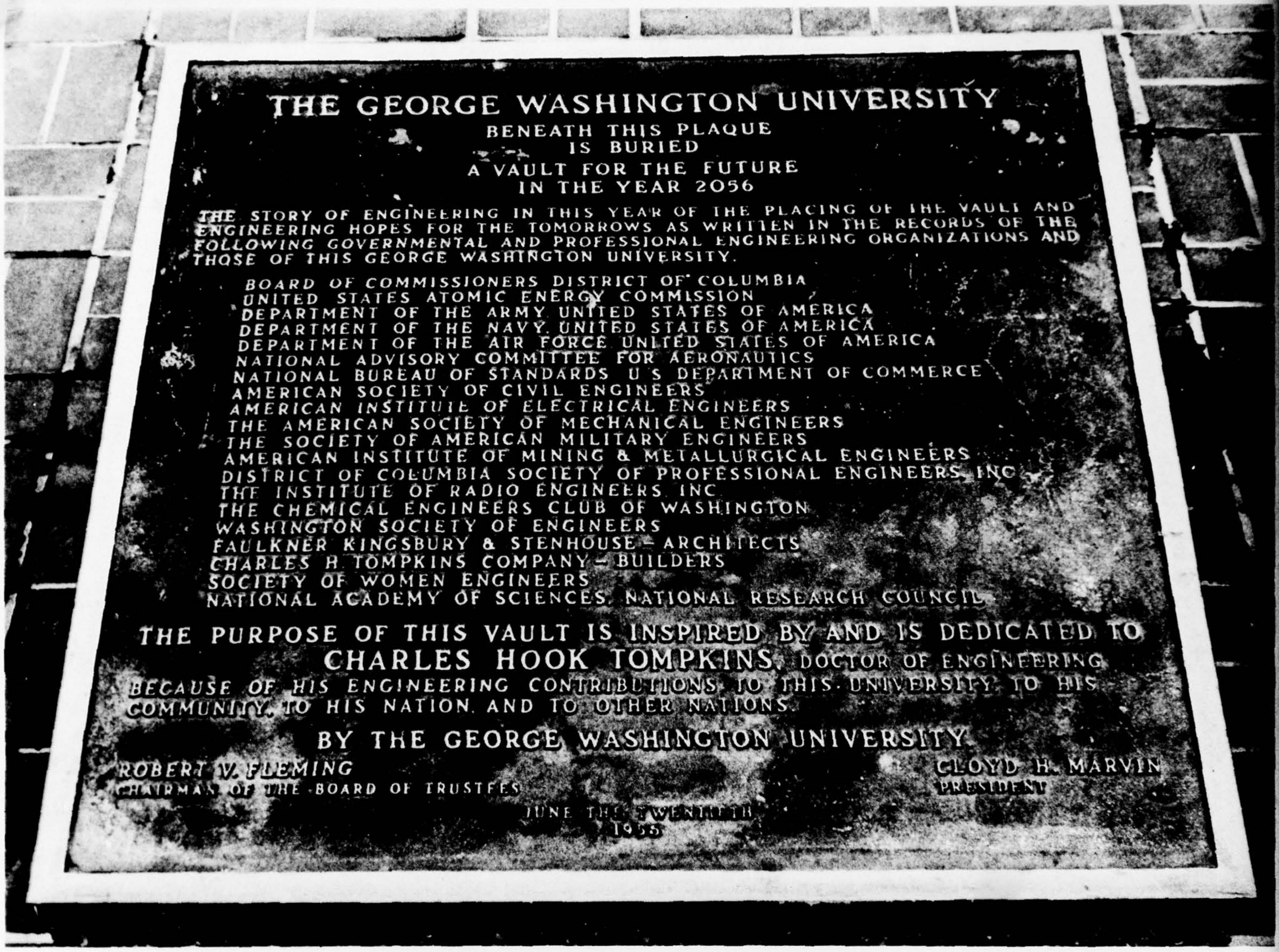
```
    BR  Ø  Ø  lab

        1. Call Write_Regs
             (see explanation on routine Write_Regs)

        2. Generate a BR register quad.
             (the generated register is
             BR  Ø  Ø  lab)
```

5) Label R QUAD

```
    LAB  lab  Ø  Ø

        1. Call Write_Regs.
             (see explanation on routine Write_Regs)

        2. Delete all variables assigned to all registers.

        3. Set STATUS entry of all registers to 'FREE'.

        4. Generate a LAB register quad.
             (generated register quad is the following
             LAB  lab  Ø  Ø)
```

6) BT/BF Quads

```
    BT  A  Ø  lab
    BF  A  Ø  lab

        1. Call Write_Regs;
             (see explanation on routine Write_Regs)

        2. Reg 1≤ Alloc_Reg_OP(A)
             (Reg 1 is a register number which variable A is assigned
             to.

        3. Generate a register quad accordingly
             (the register quad will be one of the following

             BT  Reg 1  Ø  Lab
             BF  Reg 2  Ø  Lab)

        4. Call Dealloc_Temp(A, Reg 2)
             (if A is a temporary variable then variable A will be
             deleted from list of variables assigned to Reg 2)
```



THE GEORGE WASHINGTON UNIVERSITY

BENEATH THIS PLAQUE
IS BURIED
A VAULT FOR THE FUTURE
IN THE YEAR 2056

THE STORY OF ENGINEERING IN THIS YEAR OF THE PLACING OF THE VAULT AND ENGINEERING HOPES FOR THE TOMORROWS AS WRITTEN IN THE RECORDS OF THE FOLLOWING GOVERNMENTAL AND PROFESSIONAL ENGINEERING ORGANIZATIONS AND THOSE OF THIS GEORGE WASHINGTON UNIVERSITY.

BOARD OF COMMISSIONERS DISTRICT OF COLUMBIA
UNITED STATES ATOMIC ENERGY COMMISSION
DEPARTMENT OF THE ARMY UNITED STATES OF AMERICA
DEPARTMENT OF THE NAVY UNITED STATES OF AMERICA
DEPARTMENT OF THE AIR FORCE UNITED STATES OF AMERICA
NATIONAL ADVISORY COMMITTEE FOR AERONAUTICS
NATIONAL BUREAU OF STANDARDS U S DEPARTMENT OF COMMERCE
AMERICAN SOCIETY OF CIVIL ENGINEERS
AMERICAN INSTITUTE OF ELECTRICAL ENGINEERS
THE AMERICAN SOCIETY OF MECHANICAL ENGINEERS
THE SOCIETY OF AMERICAN MILITARY ENGINEERS
AMERICAN INSTITUTE OF MINING & METALLURGICAL ENGINEERS
DISTRICT OF COLUMBIA SOCIETY OF PROFESSIONAL ENGINEERS INC
THE INSTITUTE OF RADIO ENGINEERS INC
THE CHEMICAL ENGINEERS CLUB OF WASHINGTON
WASHINGTON SOCIETY OF ENGINEERS
FAULKNER KINGSBURY & STENHOUSE – ARCHITECTS
CHARLES H. TOMPKINS COMPANY – BUILDERS
SOCIETY OF WOMEN ENGINEERS
NATIONAL ACADEMY OF SCIENCES, NATIONAL RESEARCH COUNCIL

THE PURPOSE OF THIS VAULT IS INSPIRED BY AND IS DEDICATED TO
CHARLES HOOK TOMPKINS, DOCTOR OF ENGINEERING
BECAUSE OF HIS ENGINEERING CONTRIBUTIONS TO THIS UNIVERSITY, TO HIS COMMUNITY, TO HIS NATION, AND TO OTHER NATIONS.

BY THE GEORGE WASHINGTON UNIVERSITY

ROBERT V. FLEMING
CHAIRMAN OF THE BOARD OF TRUSTEES

CLOYD H. MARVIN
PRESIDENT

JUNE THE TWENTIETH
1956

To cope with the expanding technology, our society must be assured of a continuing supply of rigorously trained and educated engineers. The School of Engineering and Applied Science is completely committed to this objective.